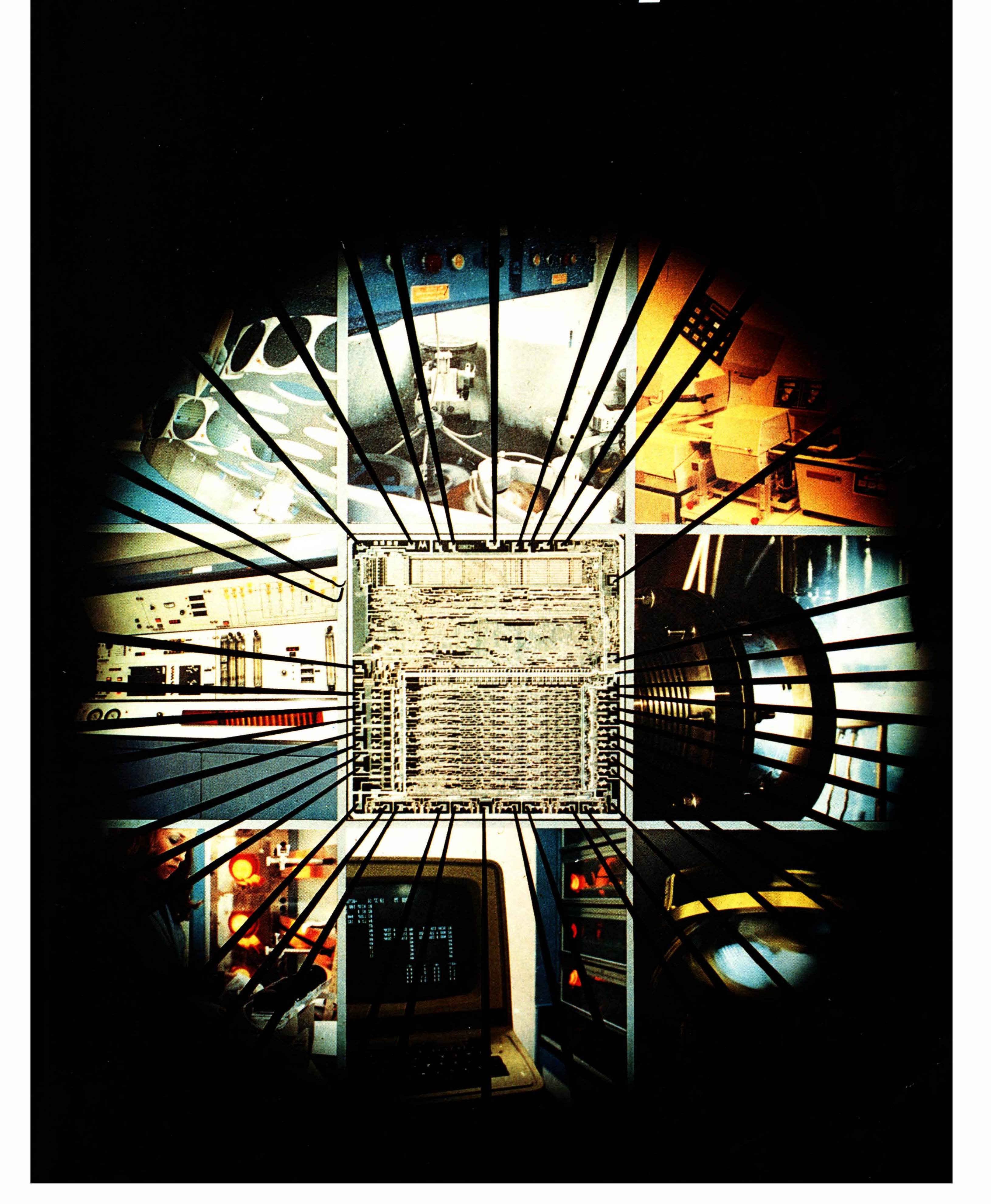
Mikrocomputer



Mikrocomputer

Mikrocomputer

Eine Zusammenfassung
aktueller Aufsätze über den Mikrocomputer
aus der Fachzeitschrift «Der Elektroniker»

Fachschriftenverlag
Aargauer Tagblatt AG, Aarau / Schweiz

Vertrieb in Deutschland:
AT-Fachverlag GmbH, Stuttgart

Separatdruck aus der Fachzeitschrift «Der Elektroniker»

Legende zum Titelbild:

Zusammenstellung verschiedener Platinherstellungsphasen bei Motorola. In der Mitte Ansicht der Mikroprozessor-Chips M6800. Die radialen Linien stellen die Anschlüsse der Platinentest-Maschine dar. (Foto Motorola)

Inhaltsverzeichnis

Mikrocomputer 9
Kurt Wüthrich, Ing.-Techn. HTL, Rüfenacht

1. Grundeigenschaften 9
2. Anwendungsbereich 10
3. Hardware 10
3.1 Technologie 10
3.2 Grundaufbau 10
3.3 Funktionsgruppen des Intel-MCS 4 11
3.3.1 Hilfsstromkreise 11
3.3.2 Speicher 12
3.3.3 I/O-Ports 12
3.3.4 CPU 12
4. Eigenschaften der Mikroprozessor-CPU 13
4.1 Überblick 13
4.2 Adressierarten 13
5. Software 13
5.1 Grundoperatoren der Programmierung 13
5.2 Instruktionstabelle 14
5.3 Programmierbeispiele 14
5.4 Mikroprogramme 15
6. Arbeitsplatz und Programmierhilfen 15
6.1 Prototypsystem 15
6.2 Assembler 15
6.3 Simulator 15
6.4 Verschiedenes 15
7. Ausblick 15

Flussdiagramm und Mikrocomputer 17
Alexander Gautschi, Autophon AG, Solothurn

1. Einleitung 17
2. Problemdefinition 17
3. Lösungsweg 18
4. Flussdiagramm 19
4.1 Symbole 19
4.2 Beispiel Kombinatorik 19
4.3 Beispiel Rechner 21
5. Zusammenfassung 22

Digitale Halbleiterspeicher 23
R. Zinniker, dipl. Ing., Institut für Elektronik, ETH, Zürich

1. Einleitung 23
2. Speicherarten, Grundbegriffe 23
2.1 RAM 23
2.2 ROM, PROM, RePROM 24
2.3 SR 24
2.4 Sondertypen 25
3. Speicherorganisation 26
4. Schaltungsarten 26
4.1 TTL 26
4.2 ECL 27
4.3 P-MOS 27
4.4 N-MOS 27
4.5 CMOS 27
5. Technologien 27
5.1 Bipolare Technologien 27
5.2 MOS-Technologien 27
6. ROM, PROM, RePROM 29
6.1 Aufbau 29
6.2 Speicherzellen 29
6.3 Anwendungen 30
7. RAM 33
7.1 Aufbau 33
7.2 Speicherzellen 34
8. Schieberegister 35
8.1 Aufbau 35
8.2 Speicherzellen 36
8.3 Anwendungen 36
9. Produkte 37

Das Programmieren von Mikrocomputern 39
Th. Kaegi, dipl. Ing. ETH, Wil

1. Das Vorgehen zum Erlernen des Programmierens 39
1.1 Kurse 39
1.2 Selbststudium 39
2. Der Aufbau eines Computers 39
2.1 Allgemeines 39
2.2 Besonderheiten des Rechners MCS-4 von Intel 39
3. Instruktionen und Daten 40
3.1 Befehlsfolge 40
3.2 Darstellungsart der Befehle 40
3.3 Binäre Zahlendarstellung 40
4. Das Instruktionsrepertoire des Rechners MCS-4 40
4.1 Befehlsliste 40
4.2 Adressieren von Datenspeicher, Ein- und Ausgabekanälen 42
5. Programmbeispiel 1 43
5.1 Aufgabe 43
5.2 Erste Lösung 43
5.3 Kritik der ersten Lösung 43
5.4 Zweite Lösung 43
6. Programmbeispiel 2 44
6.1 Aufgabenstellung 44
6.2 Das Programmbeispiel 44
7. Das Assemblerprogramm 46
8. Das Austesten 47
8.1 Simulation 47
8.2 Austesten im System 47
9. Schlussbemerkungen 47

Die Qual der Wahl beim Mikroprozessor 49
P. Schneider, Zentrallabor Landis & Gyr, Zug

1. Einführung 50
1.1 Allgemeines 50
1.2 Definitionen einiger Begriffe 50
1.3 Entwicklung der Integration 51
1.4 Versuch einer Klassifizierung von Mikro- und Minicomputern 51
2. Grundorganisationen von Computern 52
3. Mikroprogrammierung von Mikroprozessoren 52
3.1 Was ist Mikroprogrammierung? 52
3.2 Vorteile der Mikroprogrammierung für den Computerhersteller 54

3.3	Mikroprogrammierung durch den Anwender von Mikroprozessoren	54
3.4	Beispiel eines mikroprogrammierbaren Mikroprozessors (GPC/P von National Semiconductor)	55
4.	Einchip-, Multichip-Mikroprozessoren	56
4.1	Allgemeines	56
4.2	Einchip-Mikroprozessor (Computer on a chip)	56
4.3	Multichip-Mikroprozessoren	56
5.	Wortlängen von Mikroprozessoren	57
6.	Synchrone, asynchrone und pseudoasynchrone Arbeitsweise	57
7.	Parallele und zeitmultiplexe Datenübertragung	57
8.	Register und Flags	57
9.	Stack (Stapel, LIFO, Last in first out)	59
9.1	Begriff und Verwendung	59
9.2	Implementationen von Stacks bei Mikroprozessoren	59
9.3	Mikroprozessoren mit mehreren Registersätzen (MK 5065 P)	60
9.4	Probleme bei Anwendungen von ROMs bei Computern, welche die Rücksprungadresse am Anfang der Subroutine abspeichern	60
10.	Interrupts	60
10.1	Prinzip	60
10.2	Multilevel-Interrupt-Systeme	61
10.3	Reset	62
10.4	Software-Interrupt	62
11.	Input/Output-Architektur	62
12.	Instruktionsset	62
12.1	Allgemeines	62
12.2	Funktionen des Instruktionssets	62
12.2.1	Arithmetisch/logische Funktionen	62
12.2.2	Transferfunktionen	62
12.2.3	Sprungfunktionen	64
12.2.4	Verschiedene Funktionen	64
12.2.5	Kombination mehrerer Funktionen in einer Instruktion	64
13.	Elektrische Charakteristiken	64
14.	Bausteinsatz	64
15.	Technologie	64
16.	Zweitlieferant (Second Source)	65
17.	Software-Entwicklungshilfen	65
17.1	Allgemeines	65
17.2	Editor	65
17.3	Assembler	65
17.4	Compiler	66
17.5	Lader	67
17.6	Debugprogramm	67
17.7	Simulatorprogramm	67
18.	Tabellarische Übersicht der erhältlichen und angekündigten Mikrocomputer	67
18.1	4-bit-Mikroprozessoren	68
18.2	8-bit-Mikroprozessoren	69
18.3	12-bit-Mikroprozessoren	70
18.4	16-bit-Mikroprozessoren	70
18.5	Übersicht «Single board processors» (Naked Minis)	71
18.6	Übersicht mikroprogrammierbare Mikroprozessoren	71

Der Algorithmus 73
Thomas Kunz, Bern

Vorwort

«Der Elektroniker» hat in letzter Zeit oft über den Mikrocomputer berichtet. Die Aufsätze haben ein breites Interesse gefunden. Verschiedene Anfragen haben uns dazu bewogen, die Aufsätze in einem Sonderdruck zusammenzustellen und sie so dem Leser als Ganzes zugänglich zu machen.
Im «Elektroniker»-Leitwort, Heft 4/74, ist der Mikrocomputer wie folgt vorgestellt worden:

«Vor rund 20 Jahren hat der Transistor die Röhre abgelöst. Vor ungefähr 10 Jahren begannen die integrierten Schaltkreise das Gesicht der Elektronik neu zu gestalten. Heute steht mit dem Mikrocomputer oder Mikroprozessor dem Entwickler digitaler Steuerungen ein Baustein zur Verfügung, der die jetzt bereits klassische IC-Technik in vielen Systemen aus wirtschaftlichen Gründen verdrängen wird.
Der Mikrocomputer wird dort in einem System einspringen, wo ein minimaler Schaltkreisaufwand überschritten wird, es sich aber nicht lohnt, bereits einen Minicomputer einzusetzen. Mit dem Mikrocomputer lassen sich viele heute festverdrahtete Steuerungssysteme flexibler und wirtschaftlicher realisieren.
Wer mit der klassischen integrierten Schaltungstechnik aufgewachsen ist, muss umdenken. Der Hardwareaufwand wird kleiner, der Softwareaufwand aber wächst. Der Entwickler digitaler Steuerungssysteme wird sich in Zukunft noch mehr mit den Begriffen, der Sprache und den Methoden der Computertechnik beschäftigen müssen.»

In der vorliegenden Zusammenstellung wird über den Mikroprozessor, die Zentraleinheit (CPU) eines Mikrocomputers, und über die verschiedenen Halbleiterspeicher berichtet. Ferner findet man reine Software-Aufsätze, angefangen beim Flussdiagramm.
Die Aufsatzreihe gibt dem Anfänger einen Einblick in das Mikrocomputergebiet, bietet aber auch dem Fortgeschrittenen Anregung und praktische Hilfe, insbesondere mit der ausführlichen tabellarischen Zusammenstellung der erhältlichen und angekündigten Mikroprozessoren. Der Sonderdruck schliesst auf dem Gebiet der Mikrocomputer eine in der deutschsprachigen Fachliteratur bestehende Lücke.

Peter Stuber
(Redaktor der Fachzeitschrift «Der Elektroniker»)

Mikrocomputer

«Die in letzter Zeit von etlichen Halbleiterfirmen eingeführten Mikrocomputer versprechen umwälzende Veränderungen in der Entwicklung von Logiksystemen.»
Dies scheint in der Tat nicht bloss ein Reklame-Ausspruch der betreffenden Hersteller zu sein, sondern vielmehr die Meinung und Erfahrung all jener, die solche Mikrocomputersysteme bereits anwenden.
Da Mikrocomputer die konsequente Verkleinerung der herkömmlichen Computer sind, werden sie mit Sicherheit für kleinere oder für Teilaufgaben in Grossystemen von Software-Leuten in ihrer gewohnten Art eingesetzt werden. Diesem Anwendungsfall tragen die Hersteller vermehrt Rechnung, indem sie ihre Rechnerchips, Halbleiterspeicher, Interface- und Peripheriestromkreise usw. fertig verpackt als «Black Box» anbieten. Als wichtige Arbeitsgrundlage dazu muss ein Hilfsprogramm verfügbar sein, das, auf einem ausreichenden Minicomputer oder auf einer Grossanlage ausgeführt, die Funktionsweise des Mikrocomputers simuliert.
Der vorliegende Aufsatz soll den zweiten Anwendungsfall beleuchten: den Ersatz von Logiksystemen herkömmlicher Technik durch Mikrocomputer. Hier wird sich der frühere Hardware-Entwickler mit den ganz neuartigen Problemen auseinanderzusetzen haben. Möglicherweise kann ihm die arbeitgebende Firma keine Software-Ausbildung oder -unterstützung bei der Entwicklung der Programme geben. Er hat sich alles Nötige im «Do-it-yourself-Verfahren» anzueignen. Bei der Auswahl eines Mikrocomputersystems wird deshalb ein wichtiger Punkt sein, in welchem Ausmass der Hersteller Anleitungen, Beschreibungen, Programmierhilfen und sogar Programmbeispiele anbietet. Die Erfahrung zeigt, dass ein Elektroniker ohne Computer- und Software-Kenntnisse nach sehr kurzer Zeit (bereits nach einigen Tagen!) in der Lage ist, seine ersten Mikroprogramme zu schreiben.

1. Grundeigenschaften

Ein Logiksystem herkömmlicher Technik, das durch Verdrahten von einzelnen Gattern, Flip-Flops, Zählern usw. zustande gekommen ist, und dessen Funktionsweise nur durch Neuverdrahten änderbar ist, kann auf folgende Arten ersetzt werden:

- Durch kundenspezifizierte Hybrid- oder Integrierte Schaltungen mit dem Ziel, durch Verminderung der Komponenten (Anzahl ICs, diskrete Elemente, Printplatten, Stecker, Drahtverbindungen) das System geometrisch zu verkleinern und zu verbilligen. Als Nachteil dürfte möglicherweise die endgültige Festlegung der Funktionsweise des Systems ins Gewicht fallen. Zudem ist dieses Vorgehen natürlich nur für sehr grosse Serien geeignet.
- Durch ein Mikrocomputersystem mit dem selben, oben beschriebenen Ziel. Diese Lösung ist geeignet, wenn kleinere Serien hergestellt werden und sie ist dann speziell geeignet, wenn innerhalb kleiner Stückzahlen Änderungen, Anpassungen, Erweiterungen usw. verlangt oder vorausgesehen werden.

Gegenüber der verdrahteten Logik weist der Mikrocomputer folgende Vor- und Nachteile auf:

Vorteile:

- Die Entwicklung von Hardware ist auf ein Minimum beschränkt Dieselbe Hardware kann für mehrere Projekte verschiedenster Art verwendet werden
- Änderungen können durch Umprogrammieren ohne jeglichen «Hardware-Aufwand» leicht vorgenommen werden
- Erweiterungen sind bei vorausgehender kluger Auslegung der Hardware durch Zusatzbestückungen möglich
- Die Entwicklungszeit ist vergleichsweise kurz. Kundenwünschen und der Marktlage kann schneller entsprochen werden
- Der Mikrocomputer ist – alles in allem – billiger

Nachteile

- Die – wenn auch kurze – Einführungszeit in das gewählte Mikrocomputersystem
  Die Aufwendung einer beträchtlichen Geldsumme für einen Programmierplatz (der aber gleichzeitig als Prototypsystem dienen kann), wenn kein genügend leistungsfähiger Mini- oder Grosscomputer zur Verfügung steht.

2. Anwendungsbereich

Die Praxis scheint zu bestätigen, dass ein Mikrocomputersystem mit sehr wenigen Ausnahmen überall eingesetzt werden kann. Die zwei hauptsächlichsten Ausnahmefälle sind:

- sehr kleine Systeme (einige hundert Gatterfunktionen) aus Preisgründen
- sehr schnelle Systeme, da die durchwegs angewendete MOS-Technik vergleichsweise langsam ist.

3. Hardware

3.1 Technologie

Die bedeutende Miniaturisierung ist nur möglich geworden dank der erfolgreichen Herstellung grossflächiger MOS-Strukturen. So sind z.B. auf dem 4-bit-CPU-Chip der Firma INTEL in PMOS-Technik rund 44 000 Schaltfunktionen vereinigt. Dieser 16polige IC ersetzt ca. 120 herkömmliche TTL-ICs. Seine Verlustleistung beträgt weniger als 1 W! Die Arbeitsgeschwindigkeit ist ungefähr eine Grössenordnung langsamer als mit einer TTL-Version.
MOS-Stromkreise sind empfindlicher auf statische Aufladung vor allem der Eingänge, als dies bipolare Stromkreise sind. Alle Ein- und Ausgänge der integrierten Schaltkreise sind zwar elektrisch geschützt. Trotzdem empfehlen sich gewisse Vorsichtsmassnahmen bei der Handhabung, wie sie in den Datenblättern beschrieben sind.
Einer näheren Beschreibung bedürfen die Speicherelemente, da sie im Mikrocomputer selbstverständlich eine zentrale Bedeutung erlangen.
RAM – Random Access Memory – Speicher mit wahlfreiem Zugriff – «Schreib-/Lese-Speicher».
Die Speicherstelle besteht im Falle des dynamisch betriebenen (gepulsten) RAM im Prinzip aus einer Kapazität, die periodisch geladen wird, beim statischen RAM aus gleichstromgekoppelten, Flip-Flop-ähnlichen Elementen.
ROM – Read Only Memory – Nur auslesbarer Speicher – «Festwertspeicher». Die Speichermatrix wird bei der Herstellung dieses ICs durch eine vom Kunden festgelegte Metallisationsmaske programmiert. Da in der Regel nur eine Maske verändert werden muss, ist das ROM bereits ab wenigen hundert Stück lieferzeitmässig und preislich interessant.
PROM – Programmierbares ROM. Der Anwender kann das als Standard-IC erhältliche PROM selbst programmieren. Die eingeschriebene Information ist nicht mehr löschbar. Als Speicherelemente dienen Widerstände oder Diodenstrecken, die bei der Programmierung durch momentane Überlastung unterbrochen werden.
Löschbare PROM – oder auch RePROM (wieder programmierbare PROM). Dank einem völlig zerstörungsfreien elektrischen Durchbruch werden Elektronen auf ein allseitig isoliertes («schwimmendes», floating) Gate gebracht. Der darunterliegende MOS-Kanal wird dadurch beeinflusst und die Speicherstelle logisch entsprechend interpretiert. Das Gate behält seine Ladung über Jahre hinaus. Durch Bestrahlung mit UV-Licht wird auf photoelektrischem Weg die Ladung zum Abfliessen gebracht. Die Speicherstelle ist gelöscht. Der ganze Vorgang lässt sich beliebig oft wiederholen.
Dieses Kapitel wäre nicht abgeschlossen ohne den Hinweis auf die Kompatibilität dieser MOS-Stromkreise an den Ein- und Ausgängen. Unter Verwendung von zwei Speisespannungen gelingt es recht einfach, die MOS-Strukturen TTL-kompatibel zu machen. Tatsächlich sind die uns bekannten Mikrocomputersysteme an den Ein- und Ausgängen auch TTL-mässig spezifiziert. Es fehlen jedoch leider Angaben über die mögliche Kompatibilität mit anderen Logikfamilien. So hat der Anwender selbst abzuklären, wie er z.B. die Trennstellen zu und von komplementären MOS-Schaltkreisen (CMOS, COSMOS) auszulegen hat. Unter Umständen spielen dabei Zeitprobleme eine ausschlaggebende Rolle.

3.2 Grundaufbau

Auch der Mikrocomputer besteht aus den folgenden drei Funktionsgruppen:

- Zentralrecheneinheit, Prozessor, im weiteren mit CPU (Central Processor Unit) bezeichnet.
- Speicher, in Mikrocomputersystemen möglicherweise hardwaremässig aufgetrennt in Schreib-/Lese-Speicher für die laufenden Daten und in Festwertspeicher für die Mikroprogramme und für Konstanten.
- Ein-/Ausgangsstromkreise, im weiteren mit I/O-Port (Input/Output-Port) bezeichnet.

Selbstverständlich gehören etliche Interface- und Hilfsstromkreise zum System. Eine mehradrige Leitung, die in beiden Richtungen benützt wird (Datenbus), verbindet auf einfachste Weise die verschiedenen Funktionsgruppen miteinander. Die Bilder 1 und 2 zeigen als Beispiel ein 4-bit-Mikrocomputersystem in einem groben und einem detaillierteren Blockschema. Anhand dieses Beispieles sollen die einzelnen Funktionsgruppen näher erläutert werden. Es sei ausdrücklich darauf hingewiesen, dass für jedes andere System das Besprochene in vielleicht etwas abgewandelter Form ebenso gültig ist. Als Beispiel ist das INTEL MCS-4-System gewählt worden, da es als in diesem Sinne erstes Mikrocomputersystem vor nahezu zwei Jahren auf dem Weltmarkt erschien und seither weite Verbreitung gefunden hat.

Die grundsätzliche Funktionsweise ist recht einfach: gemäss Bild 3 wird der Stand des Programmschrittzählers in der CPU als Programmspeicheradresse ausgesendet. Der so angewählte Speicherinhalt wird als nächste auszuführende Instruktion über denselben Datenbus in den Instruktionsdecoder in der CPU eingelesen. Je nach Instruktion arbeitet im folgenden nur die CPU (Maschinenbefehle, Accumulatorbefehle) oder die CPU zusammen mit weiteren Funktionsgruppen (RAM- und I/O-Befehle). Nach der Ausführung dieser zuletzt eingeholten Instruktion wird der Programmschrittzähler um einen Schritt aufgezählt und der nächste Maschinenzyklus in der gleichen Weise begonnen.
Eine erste Erschwerung sind die Doppelwort- (in anderen Systemen bis zu Dreiwort-)Instruktionen. Besonders bei Sprungbefehlen wird so die Möglichkeit geschaffen, dem eigentlichen Maschinenbefehl eine ausreichend lange Adresse beifügen zu können, um in einem genügend grossen Speicherbereich zu springen.

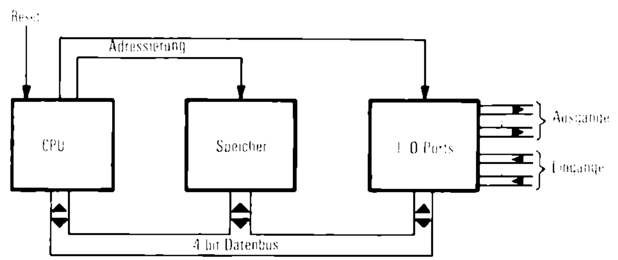


Bild 1 Blockschema eines Mikrocomputers

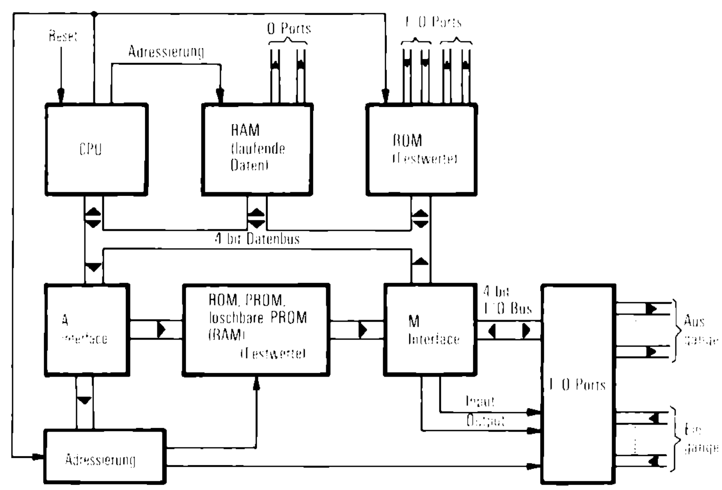


Bild 2 Verfeinertes Blockschema des INTEL-MCS-4-Mikrocomputersystems

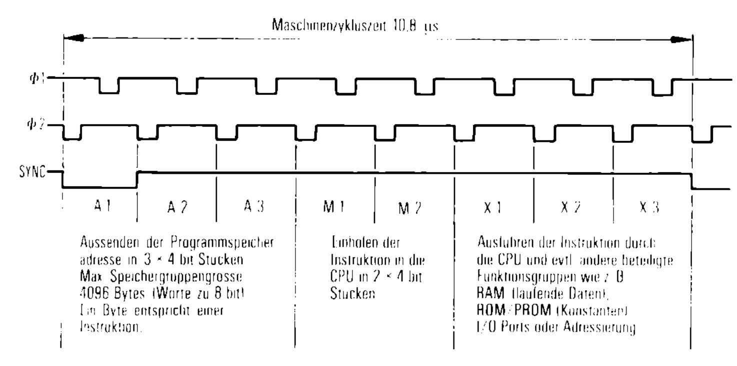


Bild 3 Timing des INTEL-MCS-4-Mikrocomputersystems

Eine weitere Erschwerung im Sinne des «Normalablaufes», aber eine absolut notwendige Eigenschaft jedes Mikrocomputers, sind die erwähnten Sprungbefehle selbst. Sie gestatten das Springen des Programmschrittzählers in praktisch beliebiger Form. Ein solcher Programmsprung kann mit einer (Doppelwort-)Instruktion direkt oder indirekt befohlen oder von einer erfüllten beziehungsweise nicht erfüllten Bedingung abhängig gemacht werden.
Die Gesamtfunktion des mikrocomputergesteuerten Systemes wird durch eine Vielzahl solcher aneinandergereihter «Mikroinstruktionen» bestimmt. Eine einfache Teilaufgabe – z. B. das Einlesen von Daten ab I-Port in einen RAM-Speicherplatz – erfordert bereits etliche Programmschritte – Adressierung und Anwahl des I-Ports, Einlesebefehl, Adressierung und Anwahl des RAM-Speicherplatzes und Einschreiben.
Eine Hauptaufgabe des Programmierers ist es, möglichst viele solcher Teilaufgaben zu «normieren» und in der Form von Subroutinen unter einem bestimmten Codewort abzuspeichern. Das Hauptprogramm schrumpft zusammen und besteht im Extremfall schliesslich nur noch aus der Erzeugung von Vorbedingungen (Setzen von Zählern, Bereitstellen von Adressen usw.) und Subroutineneinsprüngen.
Beispiel: Im Hauptprogramm wird die Adresse eines I-Ports festgelegt und ein Vergleichswort bereitgestellt. Anschliessend erfolgt der Subroutineneinsprung. Die entsprechende Subroutine beinhaltet die Anwahl des I-Ports gemäss der bereitgestellten Adresse, einlesen, MASK-Operation, indirekte Programmverzweigung in eine von mehreren möglichen (Sub) Subroutinen und nach Beendigung derselben den Rücksprung an die verbliebene Stelle im Hauptprogramm.
Diese Erläuterungen zeigen, dass wohl jeder Befehl nach genau demselben Schema A 1–A 3, M 1, M 2, X 1–X 3 (Bild 3) ausgeführt wird, dass das Programm jedoch nicht über lange Stücke «geradlinig» abläuft, sondern vielmehr Schleifen durchläuft, springt, in Subroutinen ein- und zurückspringt, verzweigt wird usw.

3.3 Funktionsgruppen des Intel-MCS 4

3.3.1 Hilfsstromkreise

A- und M-Interface

Die Arbeitsweise dieser beiden ICs ist typisch für viele Mikrocomputersysteme. Um sie genauer zu verstehen, muss man sich die zeitlichen Verhältnisse beim Programmablauf vor Augen halten (Bild 3). Da die Wortlänge in der Regel nicht ausreicht, um einen genügend grossen Speicher adressieren zu können, muss die CPU z. B. die Programmspeicheradresse in «zerstückelter» Form seriell an das A-Interface weitergeben (A_1, A_2 und A_3 in Bild 3). Das A-Interface speichert diese Datensegmente, so dass wieder eine 12-bit-Adresse in paralleler Form entsteht.
Ihm obliegt auch die Aufbereitung der I/O-Port-Adressen.
Das M-Interface hat die gegenteilige Aufgabe, indem es die 8-bit-Instruktion in 2 4-bit-Stücken während M 1 und M 2 an das CPU durchschalten muss.
Zudem übernimmt es den gesamten Datentransfer zwischen den I/O-Ports und dem CPU.

– Adressierung

Sie beschränkt sich aus den oben genannten Gründen auf einige wenige Standard-ICs (TTL- oder MOS-Gatter und Decoder).

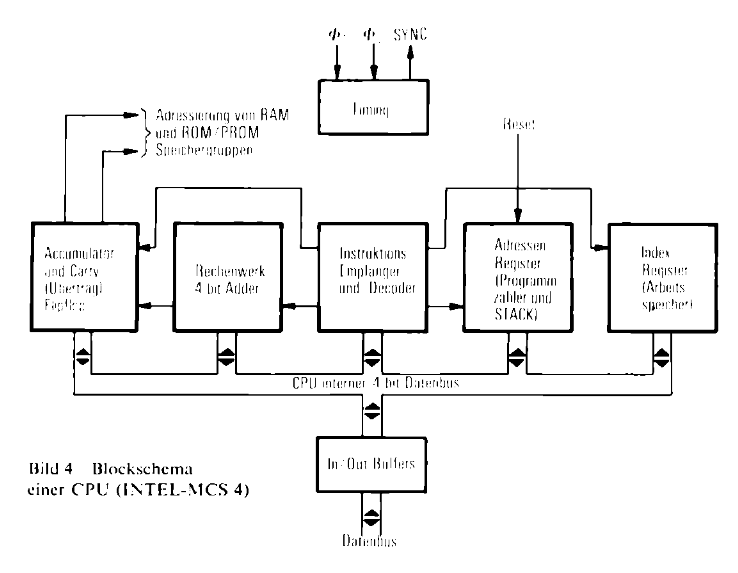


Bild 4 Blockschema einer CPU (INTEL-MCS 4)

Taktgeber

Vielfach arbeiten Mikrocomputersysteme mit einem 2-Phasen-Takt, der separat erzeugt werden muss. An die Treiberstufen werden hohe Anforderungen in bezug auf die Flankensteilheit gestellt.

– *Andere Hilfsstromkreise*

Hierunter fallen insbesondere Interrupt-Einrichtungen, d. h. Stromkreise zur Aufbereitung externer Signale, so dass der Programmablauf in bestimmten Zeitpunkten unterbrochen und ein spezielles Teilprogramm ausgeführt werden kann. Nach Ausführung dieses Teilprogrammes wird an der verbliebenen Stelle im unterbrochenen Programm bis zum nächsten Interrupt weitergefahren.
Das INTEL-MCS-4-System besitzt bloss einen System-Reset. Mit diesem Signal können sämtliche RAM-Speicher, O-Ports, Accumulatoren und Carry-Flip-Flops sowie der Programmzähler gelöscht oder auf Null gesetzt werden. Interrupt-Eigenschaften müssen unter Zeitverlust im Programmablauf softwaremässig «ersetzt» werden.

3.3.2 Speicher

Zwischen den A- und M-Interface können beliebige Speichergebilde aufgebaut werden. Von Standard-ICs in TTL- oder MOS-Technik bis zu Steckverteilern ist alles möglich! Die beiden Interface erlauben sogar einen RAM-Speicher, der mit einem Hilfsprogramm geladen werden kann.
Zusätzlich sind passend zum INTEL-MCS-4-Baustein RAM- und ROM-Speicher verfügbar, die alle nötigen Interfaceschaltungen mitintegriert haben und direkt via Datenbus und Adressenleitungen an der CPU angeschlossen werden können. Um die noch freien IC-Anschlüsse auch ausnützen zu können, enthalten beide Chips zusätzlich einen I- oder O-Port.

3.3.3 I/O-Ports

Die O-Ports bestehen je aus einem 4-bit-Latch, die I-Ports je aus einem 4-bit-Multiplexer mit Tristateausgängen, da alle Latch und Multiplexer am I/O-Bus angeschlossen sind. Auch hier kann TTL- oder unter Beachtung gewisser Regeln MOS-Technik gewählt werden.

3.3.4 CPU

Die CPU ist der komplizierteste Teil des Mikrocomputers. Es würde den Rahmen dieses Aufsatzes sprengen, näher auf die genaue Funktionsweise einzutreten. Das grobe Blockschema Bild 4

Tabelle 1 Eigenschaften von Mikrocomputer-CPUs

	INTEL MCS 4	Rockwell PPS	INTEL MCS 8	INTEL 8080	Signetics PIP	National GPC/P	AMI 7300
Wortlänge in bit	4	4	8	8	8	8, 16	8
Instruktionszeit in µs	10,8...21,6	5...10	7,5...22,5	2...6	5...10	3,3...9,6	4...32
Speichergrösse	4096 Bytes Progr./Konst. 2560 Worte (4 bit) Daten	16384 Bytes Progr./Konst. 8192 Worte (4 bit) Daten	16384 Bytes	65536 Bytes	8192 Bytes	65536 Bytes	4096 Worte Mikroprogramm und 65536 Bytes
Anzahl Instruktionen	45	54	48	78	64	42 und Mikroprogramm	Mikroprogramm
Interrupt-Eigenschaften	Reset auf Programmzähler- stand 0	Keine	Sprung auf 1 von 8 Programmstellen 1 Level	Sprung auf 1 von 8 Programmstellen Multilevel	Stack zur Speicherung des Maschinen- zustandes 1 Level	Stack zur Speicherung des Maschinen- zustandes 1/2 Level	3 Level
Adressierarten	Pointer Indirekt Immediate Register	Pointer Immediate	Pointer Immediate Register	Pointer Immediate Register	Direkt Indirekt Immediate Register Indexiert Relativ	Direkt Indirekt Immediate Register Indexiert Relativ	Direkt Indirekt Immediate Register Indexiert Relativ
Register	16 × 4 bit Programmzähler Stack (3)	2 × 4 bit Programmzähler Stack (2) Pointer (1)	5 × 8 bit Programmzähler Stack (7) Pointer (1)	5 × 8 bit Programmzähler Stack (unlimitiert) Pointer (1)	4 × 8 bit Programmzähler Stack (7)	4 × 16 bit Programmzähler Stack (16)	16 × 8 bit Programmzähler 32 × 8 Stack

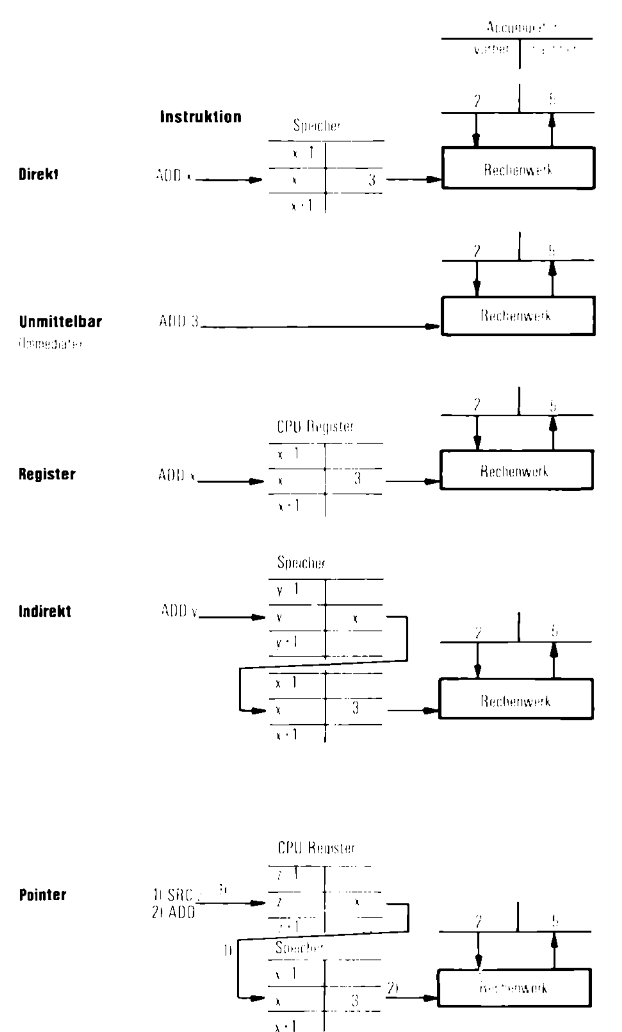


Bild 5 Adressierarten

und die Hinweise in den nächsten Kapiteln müssen hier genügen. Die eigentliche «Verknüpfungsstelle» von Hardware und Software ist der Instruktionsdecoder, der «Arbeitsverteiler» im System.
Maschinenbefehle berühren Rechenwerk, Adressenregister und Indexregister. Für die Accumulatorgruppe wird eine eigene Instruktionstabelle gebildet. Die dritte Instruktionsgruppe umfasst alle Befehle für Speicher und I/O-Ports. Deren Ausführung verlangt eine Zusammenarbeit der CPU mit der entsprechenden Funktionsgruppe.

4. Eigenschaften der Mikroprozessor-CPU

4.1 Überblick

Die hauptsächlichsten Eigenschaften heutiger Mikroprozessoren sind Wortlänge, Arbeitsgeschwindigkeit, Adressierarten, Interrupt-Möglichkeiten, Anzahl Register und – möglicherweise nicht zuletzt – der Preis. Tabelle 1 gibt einen Überblick über einige heute oder in naher Zukunft erhältliche CPU.
Anzahl und Art der Instruktionen sind unter Umständen ebenso wichtige Auswahlkriterien. So findet man z. B. in der Instruktionstabelle des INTEL-MCS 4 einige Befehle besonders für die Verarbeitung von BCD-Daten, während diejenige des MCS 8 etliche Befehle für logische Verknüpfungen (AND, OR, EXCLUSIVE OR, COMPARE usw.) enthält. Dagegen gibt es andere Systeme mit hervorstechenden Interrupt-Eigenschaften.

Verallgemeinernd darf gesagt werden:
– je mehr interne Register und je mehr verschiedene Adressierarten ein CPU aufweist, desto schneller arbeitet das System und desto weniger externe Speicher sind nötig;
– je mehr Befehle die Instruktionstabelle umfasst, desto kürzer und übersichtlicher werden die Mikroprogramme.

4.2 Adressierarten

Bild 5 zeigt in bildlicher Darstellung Beispiele für einige Adressierarten. Ein hochwertiges Mikrocomputersystem sollte möglichst viele davon benützen.

– Direkte und Register-Adressierung sind die einfachsten und gebräuchlichsten.
– Immediate ist geeignet für konstante Werte (z. B. Setzen eines Registers, das als Zähler verwendet wird).
– Die indirekte und die «Pointer»-Adressierung sind sehr leistungsfähige «Software-Werkzeuge», indem sie einen direkten Speicherzugriff zulassen, wie sie die kurze Wortlänge der meisten Mikrocomputer verbieten würde. Als Beispiel: INTEL-MCS 4. Die Wortlänge von nur 4 bit erlaubt eine direkte Adressierung von 16 Speicherplätzen. Die «Pointer-Register» umfassen jedoch 8 bit, so dass ein direkter Zugriff zu 256 Speicherplätzen geschaffen wird.

Direkte, Register- und Immediate-Adressen sind ausnahmslos in der auszuführenden Instruktion selbst enthalten. Jede Instruktion besteht aus einem Operator und aus einem Operanden. Der Operator bestimmt, was zu tun ist, währenddem der Operand festlegt, auf welchen Daten diese Operation zu erfolgen hat.

5. Software

5.1 Grundoperatoren der Programmierung

So wie für die verdrahtete Logik schliesslich AND, OR und NOT die Grundfunktionen darstellen, gibt es auch in der Programmierung drei Grundoperatoren:

– MASK: Prüfung einzelner Bits in einem Wort. Die nicht geprüften Bits können einen beliebigen Zustand annehmen.
– COMPARE: Prüfung eines ganzen Wortes, d. h. Vergleich mit einem vorgegebenen Bitmuster.
– Jump: Sprung an diejenige Programmstelle, die als Resultat einer MASK- oder COMPARE-Operation den gewünschten Ablauf bringt.

Auch wenn ein bestimmter Mikroprozessor nicht unbedingt solche Instruktionen in seiner zugehörigen Tabelle aufweist, sollte ein Äquivalent dazu aber in irgendeiner Form möglich sein. Mit den logischen Operatoren AND, OR und EXCLUSIVE OR, die nicht mit den entsprechenden Hardwarefunktionen zu verwechseln sind (!), können MASK und COMPARE am einfachsten ersetzt werden. Tabelle 2 zeigt die Arbeitsweise dieser drei Logikoperatoren.

Tabelle 2 Arbeitsweise der logischen Operatoren

Zu prüfendes Bitmuster		1000 1101	1x1x	1110
Vergleichswort		1010 1010	1010	1110
AND	Accumulator	1000 1000	*1x1x*	1110
OR	Accumulator	1010 1111	1x1x	1110
EXCLUSIVE OR	Accumulator	0010 0111	0x0x	*0000*

Um eine MASK-Operation zu Ende zu führen, muss vom *-gekennzeichneten Wert 1x1x in Tabelle 2 das Vergleichswort subtrahiert werden. Ist das Resultat im Accumulator grösser oder gleich Null, so sind alle geprüften Bits logisch «1». Dagegen genügt für die COMPARE-Operation eine Prüfung des Accumulatorinhaltes 0000. Solche Prüfungen erfordern in der Regel einen oder mehrere bedingte Programmsprünge, d.h. das Programm springt an eine vorbestimmte Stelle, falls der Accumulatorinhalt z.B. Null ist. Wenn nicht, wird die nächste Instruktion ausgeführt.
Man darf sich nicht täuschen lassen, wenn solche Operatoren in der Instruktionstabelle fehlen. Mit Sicherheit findet der Programmierer einen gleichwertigen Ersatz, der unter Ausnützung ganz anderer Instruktionen diese drei Grundoperationen erfüllt. Ein ausgezeichnetes Beispiel dafür ist im übernächsten Kapitel unter der Überschrift «Subroutineneinsprung» beachtenswert.

5.2 Instruktionstabelle

Je nachdem, wie ein Mikrocomputersystem und seine CPU im Detail hardwaremässig organisiert sind, unterscheidet man gerne mehrere Befehlsgruppen. Für das INTEL-MCS-4-System kann – wie bereits angetönt – wie folgt gruppiert werden:

Maschinenbefehle:

- Index-Register: Laden, Aufzählen, Aufzählen mit «Überlaufprüfung», Austausch mit Accumulatorinhalt usw.
- Rechenwerk: Arithmetik, logische Operationen
- Programmzähler und STACK: Direkte, indirekte und bedingte Programm- und Subroutineneinsprünge, Interrupt-Behandlung usw.
- Adressierung: Pointer-Register, indirekte Adressierungen

Accumulatorgruppenbefehle

- Accumulator: Aufzählen, Abzählen, Laden, Schieben, Komplementieren usw.
- Flip-Flops für spezielle Konditionen (CARRY-Übertrag des Rechenwerkes): Setzen, Löschen, Komplementieren

Speicher- und I/O-Port-Befehle:

- Speicher: Einschreiben oder auslesen in/aus Accumulator oder CPU-Register, direkte Addition ins Rechenwerk usw.
- I/O-Ports: Einlesen und Ausgeben von Daten in/aus Accumulator oder über direkte Datenkanäle (DMA) in/aus dem Speicher usw.

Sobald die Hardware des Mikrocomputers und der an ihm angeschlossenen, zu steuernden Einheiten festliegt, verbleibt «nur» noch die Vielzahl der verschiedenen Instruktionen zur Erfüllung der geforderten Arbeitsweise des Gesamtsystems. Anfänglich kann dies dem früheren Hardware-Entwickler Mühe bereiten. Denn um die Programmierbarkeit des Systems als *den* Vorteil zu erkennen, muss er sämtliche Befehle und Programmiermöglichkeiten seines Mikrocomputers genauestens beherrschen. Und auch hier hat man nie ausgelernt! Es ist deshalb empfehlenswert, in der Anlernphase genügend Übungsprogramme zu schreiben, und vielleicht auch gleiche Teilprobleme auf verschiedene Arten zu programmieren. Man wird dabei sehen, dass es unter Umständen einfacher ist, z.B. Gatterfunktionen in den zu steuernden Einheiten selbst zu erfüllen. Andererseits können durch den «sturen» Programmablauf zeitliche Probleme elegant beiseite geschafft werden.

5.3 Programmierbeispiele
(INTEL-MCS-4-Terminologie)

Verzögerungs-Subroutine (Ansteuerung von Anzeigen, elektromechanischen Elementen, Tastenentprellung usw.)

V, FIM OP, 12: V steht als symbolische Bezeichnung der Programmstelle für den Subroutineneinsprung. FIM veranlasst das Laden eines CPU-Indexregisters, OP bezeichnet das zu «bearbeitende» Index(doppel)register, und 12 ist der zu setzende Wert.
- A, ISZ 0, A: Aufzählen des Indexregisters 0. Wenn kein Überlauf stattfindet, springt das Programm nach A zurück. Da mit dem vorderen Befehl das nullte Paar, bestehend aus den beiden Registern 0 und 1 mit 12 geladen worden ist, muss dieser ISZ-Befehl 16mal hintereinander ausgeführt werden, bis dass ein Überlauf, d.h. die erneute Stellung 0 des Registers erreicht ist.
- ISZ 1, A: Das Register 1 ist ursprünglich mit 12 geladen. Es muss also um 4 aufgezählt werden. Somit ergibt sich eine Gesamtverzögerung von $4 \times 16 + 4$ Instruktionen. Da ISZ eine Doppelwortinstruktion ist, entspricht dies einer Zeit von rund $68 \times 21{,}6$ µs = 1,87 ms. (Durch Aufzählen aller 8 Registerpaare liesse sich theoretisch eine Verzögerungszeit von 15 Millionen Jahren erreichen!)
- BBL X: Am Ende dieser Subroutine wird mit diesem Befehl ein Rücksprung an die verbliebene Stelle im Hauptprogramm verlangt. Dabei wird gleichzeitig der Accumulator mit einem Wert x = 0...15 geladen.

Subroutineneinsprung, durch Anlegen von Daten an einem I-Port indirekt adressiert. Eine solche Routine ersetzt die MASK-Operation, und sie könnte auch – allerdings mit einer beträchtlichen Zeiteinbusse – Ersatz für die beim MCS 4 fehlenden Interrupt-Eigenschaften sein.
- LDM 5: Laden des Accumulators mit 5.
- XCH 2: Austausch des Accumulatorinhaltes mit dem des Index-Registers 2. In diesem steht nun 5.
- SRC 1 P: Der Inhalt des Registerpaares 1 (umfassend die beiden Register 2 und 3) wird als «Pointer»-Adresse ausgesendet und wählt den I-Port 5 an. Dieser bleibt nun so lange angewählt, bis ein neuer SRC-Befehl ausgeführt wird. (Diese Pointer-Adressierung ist anfänglich eine grosse Fehlerquelle.)
- RDR: Die am I-Port 5 anstehenden Daten werden in den Accumulator eingelesen.
- XCH 4: Die eingelesenen Daten werden in Register 4 übertragen.
- JIN 2 P: Die im Registerpaar 2 (umfassend die beiden Register 4 und 5) stehende indirekte Adresse wird in den Programmzähler übertragen. Sie ist ein Vielfaches von 16 (das da Register 5 Null ist), und der Sprung erfolgt somit auf eine von 16 Subroutinen à je 16 Programmschritte innerhalb eines Speicherbereiches von 256. Nach der Beendigung dieser Subroutine mit BBL X wird die nächste auf JIN 2P folgende Instruktion ausgeführt.

3-*Weg-Vergleich* (grösser, gleich, kleiner) COMPARE-Funktion
- LDM 5, XCH 0: Das Index-Register 0 wird mit 5 geladen. 5 ist das Vergleichswort (0101).
- RDM: Einlesen des mit dem letzten SRC-Befehl angewählten 4-bit-Wortes aus dem RAM in den Accumulator.
- CLC: Löschen des CARRY(Übertrag)-Flip-Flops
- SUB 0: Subtrahieren des Registerinhaltes 0 vom Accumulator. Stimmt das geprüfte Wort mit dem Vergleichswort überein, so ist das Resultat dieser Subtraktion Null.

- JAZ EQ: (*J*ump if *A*ccu is *Z*ero). Bei Gleichheit erfolgt ein bedingter Programmsprung nach EQ.
- JCO POS: (*J*ump if *C*arry is *O*ne). Ist der eingelesene Wert grösser als 5, erfolgt der bedingte Sprung nach POS.
- JUN NEG: (*J*ump *UN*conditional). Der eingelesene Wert ist kleiner als 5, es erfolgt ein bedingungsloser Sprung nach NEG.

5.4 Mikroprogramme

Die meisten Hersteller von Mikrocomputern bieten fertige Programme allenfalls unter dem Titel «Firmware» an. Firmware ist hardwaremässig greifbare Software. Es sind in der Regel programmierte ROMs oder PROMs, die Standardprogramme wie Routinen für die vier mathematischen Grundoperationen, Testprogramme für die CPUs, Interface-Programme für Tastenfelder, Teletype (Fernschreibterminal) usw. enthalten. Zum Teil können ebenfalls Programme, wie sie in Kapitel 6 beschrieben werden, als Firmware gelten. Sie sind der unerlässliche Software-Grundstock des Systems, der es erst überhaupt erlaubt, sinnvoll zu arbeiten.
Mit den Ausdrücken «Mikroprogramm» und «mikroprogrammiert» präzisiert man die Art des (in ROM oder PROM) abgespeicherten Programmes:
- Eine Reihenfolge von Instruktionen, die jede für sich die Funktion des Mikroprozessors während jedem Programmschritt festlegt.

6. Arbeitsplatz und Programmierhilfen

6.1 Prototypsystem

Das Herzstück des Arbeitsplatzes ist der Mikrocomputer selbst. Ohne die zusätzlichen Eigenschaften des Arbeitsplatzes (voll) auszunützen, wird er durch das Laden von selbst entwickelten Programmen und mit den angeschlossenen, zu steuernden Peripherieeinheiten als Prototypsystem benützt. Das Programm kann z.B. ab Lochstreifen in RAMs eingelesen oder in der Form von programmierten (löschbaren) PROMs «eingesteckt» werden. Im ersten Fall benötigt man zusätzlich die Interface zum Anschluss eines Teletype oder Lochstreifenlesers sowie natürlich ein entsprechendes Hilfsprogramm, das z.B. als Mikroprogramm gekauft werden kann. Im zweiten Fall wäre es zumindest wünschenswert, wenn am selben Arbeitsplatz auch die PROMs programmiert werden könnten. Dies ist möglich mit einer am Mikrocomputer angeschlossenen und durch ein separates Programm gesteuerten Programmiereinrichtung.

6.2 Assembler

Die für die Programmierarbeit wichtigsten Zusätze sind hardwaremässig die Anschlussmöglichkeit für eine Dateneingabe-/-ausgabestation (z.B. Teletype, 8-bit-Fernschreibterminal mit Tastatur, Drucker, Lochstreifenstanzer und Lochstreifenleser) und softwaremässig das sogenannte Assembler-Programm, das ebenfalls als Mikroprogramm verfügbar sein muss.
Die Programmierung des Mikrocomputers in seiner «Maschinensprache» (1-0-Sprache) ist wohl möglich, aber – gelinde ausgedrückt – unsinnig. Vor allem die Adressierungen verhelfen einem solchen Unterfangen bald einmal zum Abbruch (Ein 65 K-Speicher muss mit einer 16-bit-Adresse angewählt werden. Wieviel sind 37963 in binärer Form?). Der Assembler übersetzt das in der uns bereits bekannten «Assemblersprache» geschriebene Programm in die Maschinensprache unter Berechnung aller mit Symbolen «codierten» Adressen (JUN A – Sprung nach A). Sämtliche Konstanten werden in dezimaler Form angenommen.
Im Gegensatz zum Assembler ist der Cross Assembler in einer problemorientierten Computersprache wie zum Beispiel Fortran IV geschrieben. Das Cross-Assembler-Programm gelangt auf einer Grossanlage zur Ausführung. In beiden Fällen ist das Resultat z.B. ein Lochstreifen mit dem ursprünglichen Programm nun in der Maschinensprache, der zur Eingabe des Programmes in einen Simulator, in einen RAM-Programmspeicher oder zur Programmierung von (löschbaren) PROMs dient.

6.3 Simulator

Das Simulatorprogramm dient dem «dynamischen» Test der selbst entwickelten Programme. Als Mikroprogramm wird es auf dem Arbeitsplatz selbst ausgeführt. Das zu prüfende Programm wird in RAM-Speicher eingelesen und anschliessend Schritt für Schritt simuliert, d.h. dass an jeder beliebigen Programmstelle sämtliche Bedingungen (Registerinhalte, Konstanten, Programmzähler, STACK usw.) geprüft und/oder verändert werden können. Das Kommunikationsmittel ist der Teletype, mit dem auch alle Eingaben (Lesen von I-Ports) getätigt werden. Selbstverständlich muss man sich dabei im klaren sein, welche Vorgänge z.B. an O-Ports ausgeschriebene Daten in den angeschlossenen Peripherieeinheiten im Betrieb auslösen werden. Der Simulator ist eine schnelle und die bequemste Testmöglichkeit und nicht selten der einzige Weg, besondere Massnahmen in der Programmierung zuverlässig auf ihre Funktionstüchtigkeit auszuprüfen.
Simulatorprogramme gibt es auch z.B. in Fortran IV. Die Möglichkeiten einer Computer-Grossanlage bringen auch erweiterte Fähigkeiten eines solchen Simulatorprogrammes.

6.4 Verschiedenes

Das beste und ausgeklügeltste Mikrocomputersystem gelangt nur schwer in eine engere Auswahl, wenn der Arbeitsplatz dazu unvollständig oder unzweckmässig ist. In den seltensten Fällen wird der frühere Hardware-Entwickler über eine Computeranlage verfügen können. Er benötigt Assembler, Simulator- und allenfalls PROM-Programmierprogramme in der Maschinensprache, die mit dem eigenen System zusammenarbeiten. Man darf füglich behaupten, dass auch in dieser Beziehung die Firma INTEL bahnbrechend gewesen ist. Heute sind Programmierplätze verfügbar, die in wenigen Minuten zusammengestellt, angeschlossen und zum Einschalten bereit sind.

7. Ausblick

Mikrocomputer beweisen täglich in vermehrtem Masse ihre Vielseitigkeit und Zuverlässigkeit. Durch die MOS-Technologie und damit verbunden die drastische Reduzierung der Komponenten konnte die Ausfallquote gesenkt, das System sicherer gegenüber äusseren Störeinflüssen, kleiner und billiger gemacht werden. Das mikroprozessorgesteuerte Gerät kann auch nach seiner Ablieferung «im Felde abgeändert» werden. Hardware-Detailfragen sind auf ein Minimum beschränkt.
PROM, CPU, STACK, ASSEMBLER – alte und zum Teil längstbekannte Ausdrücke für eine neue Technik. Sie werden dem Neuanwender von Mikrocomputern bald geläufig sein.

Flussdiagramm und Mikrocomputer

Die stürmische Entwicklung auf dem Halbleitermarkt zwingt den Schaltungstechniker der Gegenwart, möglichst schnell mitzudenken. Es dreht sich dabei im speziellen um den Mikroprozessor (MP). Der Schritt von der bisherigen Digitalelektronik mit mehr oder weniger komplexen integrierten Schaltkreisen zum Mikroprozessor ist vergleichbar mit dem Sprung vom Relais zum Halbleiter. Er erfordert ein völliges Umdenken. Wenn wir nämlich ein Problem, welches bisher mit logischen Elementen gelöst wurde, direkt auf einen MP übersetzen, so wird das vielleicht funktionieren, aber es ist keinesfalls optimal. Wie man zu einer vernünftigen Lösung kommen kann, soll im folgenden erklärt werden.

1. Einleitung

Der Aufbau eines MP-Systems wurde im vorhergehenden Aufsatz beschrieben und auch seine Funktionsweise erklärt. Die Schwierigkeit für den heutigen Entwickler dürfte in der Programmierung liegen. Zuerst sind die verschiedenen Programmphasen festzuhalten:

1. Genaue Problemdefinition
2. Bestimmung des günstigsten Lösungsweges
3. Entwicklung eines Flussdiagrammes
4. Übersetzen in die Programmiersprache
5. Programmtest

Im folgenden Artikel sollen die Punkte 1 bis 3 behandelt werden.

2. Problemdefinition

Die Hauptarbeit wird bei der genauen Problemdefinition liegen. Die Ein- und Ausgangsgrössen sind in den richtigen logischen und zeitlichen Ablauf zu stellen. Zuerst wird also eine Liste derjenigen Kriterien entstehen, welche den Problemkreis beeinflussen. Als nächstes werden die Grössen zusammengetragen, die ein-, aus-, umgeschaltet oder sonst irgendwie beeinflusst werden müssen. Zuletzt müssen die beiden Listen miteinander verknüpft werden. Das Vorgehen soll an einem Beispiel aus der kombinatorischen Logik gezeigt werden.

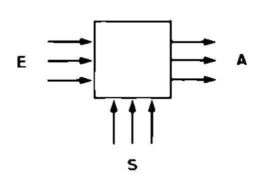


Bild 1 Problemstellung: E Eingang, A Ausgang, S Steuerleitung

Wir haben z.B. Eingänge E, Ausgänge A und Steuereingänge S. Die Steuerleitungen geben Auskunft über die Verknüpfungsart (Bild 1).

Das Beispiel ist sehr allgemein gehalten, kann aber doch auf verschiedene Gebiete sinngemäss angewendet werden, wie Verkehrssteuerung, Verfahrenstechnik, Automation in Fabrikationsbetrieben usw. Auf die Verkehrssteuerung angewendet, bedeuten die

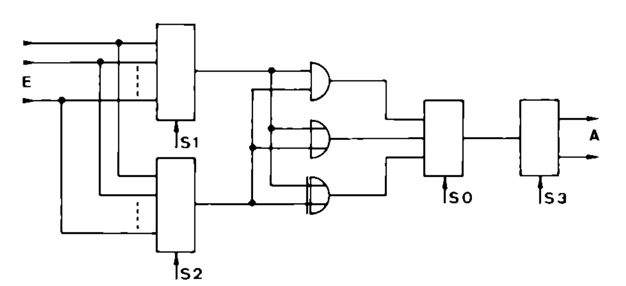


Bild 2 Hardware-Lösung. Zur Definition der Ein- und Ausgänge sowie der Verknüpfungsart werden Multi- und Demultiplexer verwendet.

Eingänge Verkehrsflusszähler, welche in den verschiedenen Zubringerstrassen stationiert sind. Die Ausgänge entsprechen den Lichtsignalen und Richtgeschwindigkeitsanzeigen. Mit den Steuergrössen werden die Zähler nach vorbestimmten Programmen ausgewertet und steuern unter Einbezug der Tageszeit die Signalanlagen.

Zum besseren Verständnis verlassen wir dieses doch recht komplexe Problem und konstruieren uns ein einfacheres nach obigem Schema (Bild 2):

$E_1 \ldots E_{10}$ sind Werte mit je 4 bit

$A_1 \ldots A_2$ sind Resultate mit je 4 bit

S_0 bestimmt die Art der Verknüpfung (wenn wir S_0 auch mit 4 bit annehmen, haben wir demzufolge 16 verschiedene Arten zur Verfügung)

$S_1 \ldots S_2$ bestimmen, welche Eingänge verarbeitet werden sollen

S_3 bezeichnet den Ausgang

Um die Phase der Problemdefinition korrekt abzuschliessen, werden wir für unser Beispiel noch einige konkrete Verknüpfungsarten bestimmen: im folgenden soll gelten:

S_0 = 0000 keine Operation
= 0001 AND
= 0010 OR
= 0100 EXOR

3. Lösungsweg

Somit kommen wir zum zweiten Schritt, nämlich zum Bestimmen des günstigsten Lösungsweges. Dieser Weg soll für irgendein Kriterium ein Optimum darstellen, das aber sehr vom Problem abhängen wird. Beim einen wird die Zeit eine Rolle spielen, d.h. wie schnell erhalte ich meine Resultate; beim andern liegt das Gewicht eher auf dem Speicherplatz, d.h. der Anzahl der verwendeten Bauteile. Hier stehen aber weder die Zeit noch die Speicheroptimierung zur Diskussion, sondern vielmehr der Unterschied zwischen Hardware- und Software-Lösung.
Bei einer Hardware-Realisierung wird man die Eingänge mit Multiplexern in Funktion der Steuerlinien auf Verknüpfungsglieder führen und davon wiederum mit Demultiplexern auf die Ausgänge verteilen (Bild 2).
Die Software-Lösung ist in unserem Fall gar nicht so sehr verschieden. Wir werden Programmblöcke bilden, welche die Verknüpfungen ausführen; die Multi- bzw. Demultiplexer erscheinen im Programm als Speicherzuweisungen. Das heisst mit anderen Worten, man sucht die Gleichartigkeiten im Problemkreis zu-

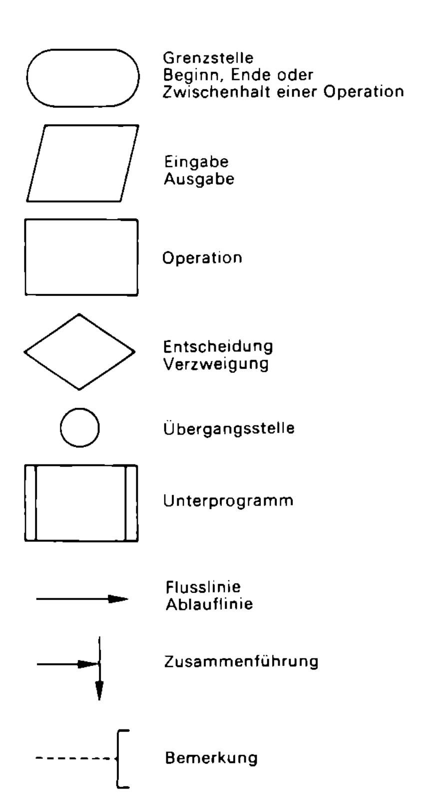


Bild 3 Flussdiagrammsymbole nach DIN 66001

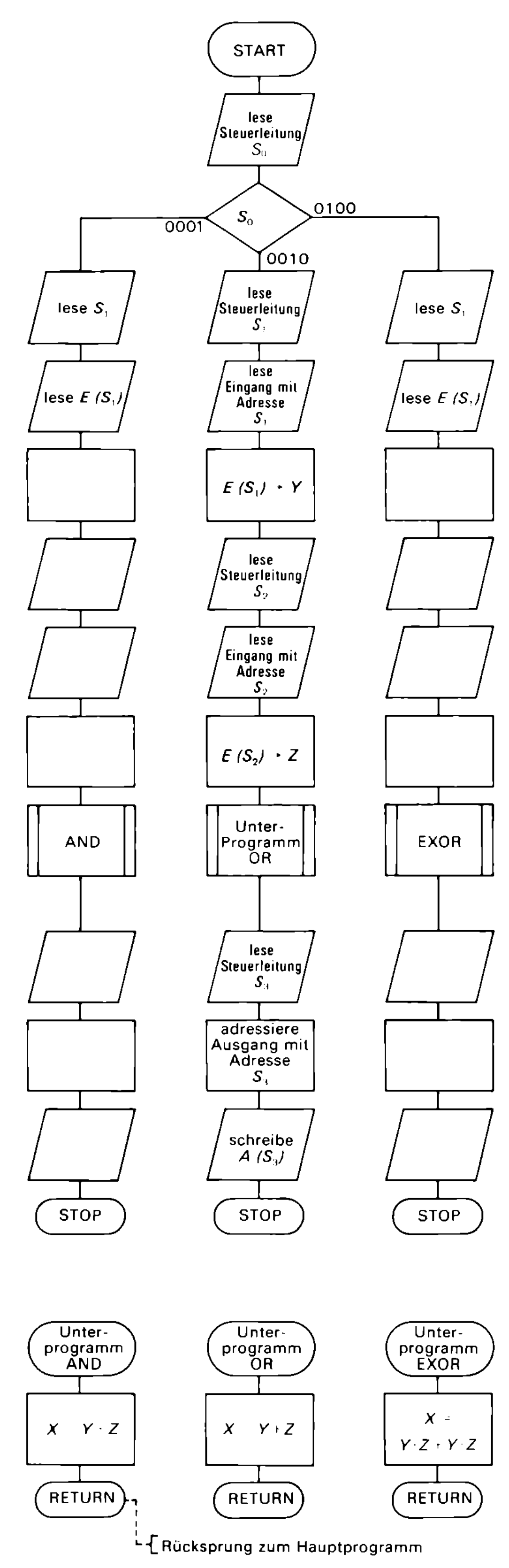


Bild 4 Flussdiagramm des kombinatorischen Beispiels

sammen. Diese werden im Programm nur einmal ausgeführt, dafür aber mit Universalgrössen. Wir haben z. B. einen Programmteil, der die Operation
$X = Y + Z$
ausführt. Wenn nun die Aufgabe $A = B + C$ und $D = E + F$ lautet, so werden wir zuerst die Zuweisung $B \rightarrow Y$ und $C \rightarrow Z$ vornehmen, die Operation ausführen und das Resultat $X \rightarrow A$ zuweisen. Dasselbe mit der zweiten Addition. Wir haben also nur einen einzigen Block benötigt. Hier zeigt sich aber auch gleich ein eminenter Unterschied zur Hardware, der vielfach ein grosses Hindernis beim reellen Einsatz darstellt. Nämlich die Tatsache, dass der MP einen Befehl nach dem andern ausführt und im allgemeinen nicht zur parallelen Verarbeitung von mehreren Grössen fähig ist. Man muss sich also im klaren sein, dass die Resultate mit einer mehr oder weniger grossen zeitlichen Verzögerung eintreffen werden.

START
lese S_1
lese $E(S_1)$
$E(S_1) \rightarrow Y$
lese S_2
lese $E(S_2)$
$E(S_2) \rightarrow Z$
lese S_0
$S_0 =$
0001
0010
0100
AND
OR
EXOR
lese S_3
$X \rightarrow A(S_3)$
schreibe $A(S_3)$
STOP

Bild 5 Vereinfachtes Flussdiagramm des kombinatorischen Beispiels

4. Flussdiagramm

4.1 Symbole

Nun kommen wir zum eigentlichen Problem, der Erstellung eines Flussdiagramms. Zuerst die Erklärung der hauptsächlich verwendeten Elemente (nach DIN 66001, Bild 3).

4.2 Beispiel Kombinatorik

Mit diesen Bauteilen wird unser Beispiel in Bild 4 dargestellt:
Wie wir gleich sehen können, ist dieses Programm sehr aufwendig geworden. Mit kleinen Umstellungen kann es wirkungsvoller werden (Bild 5).
Durch das Verschieben des S_0-Entscheides im zeitlichen Ablauf haben wir das Programm auf gut einen Drittel seiner ursprünglichen Grösse reduzieren können. In dieser Form entspricht es ziemlich genau der Hardware-Lösung nach Bild 2.
Durch Vergleichen der beiden Bilder können wir einige Zuordnungen ersehen. Zwei davon sind in Bild 6 dargestellt.

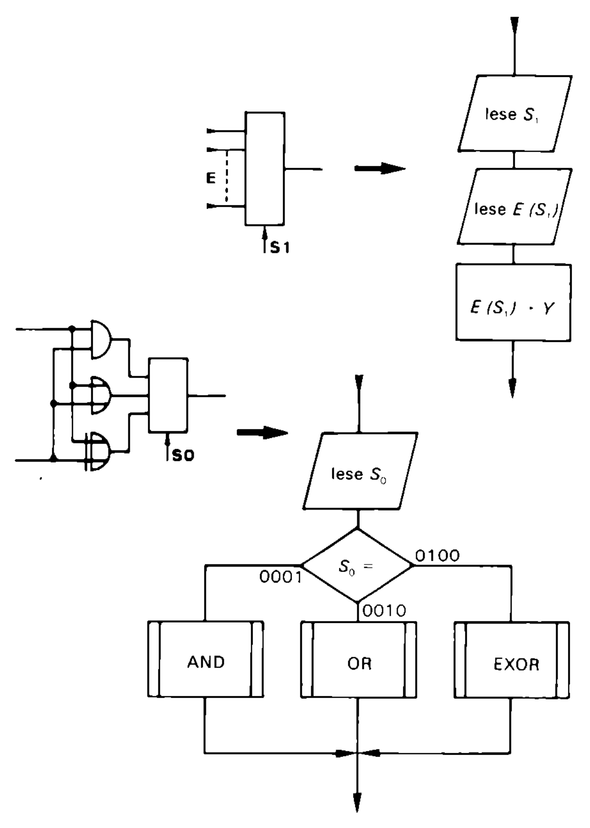


Bild 6 Zuordnung von Hardware → Software für kombinatorische Teilprobleme

Der Hardware-Anteil fällt in diesem Vergleich sehr klein aus. Wir dürfen aber nicht vergessen, dass die zu verarbeitenden Grössen aus 4 bit bestehen. Die Detaillösung wird also wesentlich komplexer aussehen.
Wenn für ein Programm noch mehrere Variable (E, S) gelesen und abgespeichert werden müssen, kann sich auch ein spezielles Lese-Unterprogramm gemäss Bild 7 lohnen.
Der Übergang vom Programm (Bild 5) zum allgemeineren Programm wird in Bild 8 gezeigt.

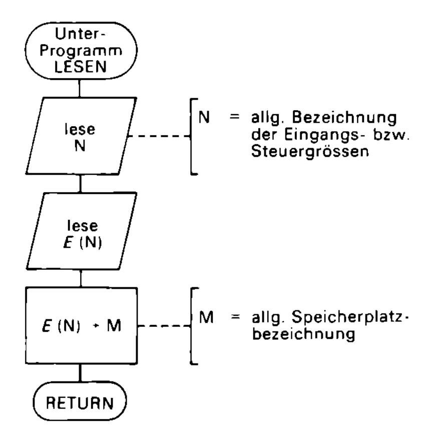


Bild 7 Unterprogramm zum Lesen und Speichern vieler Eingangs- und Steuergrössen

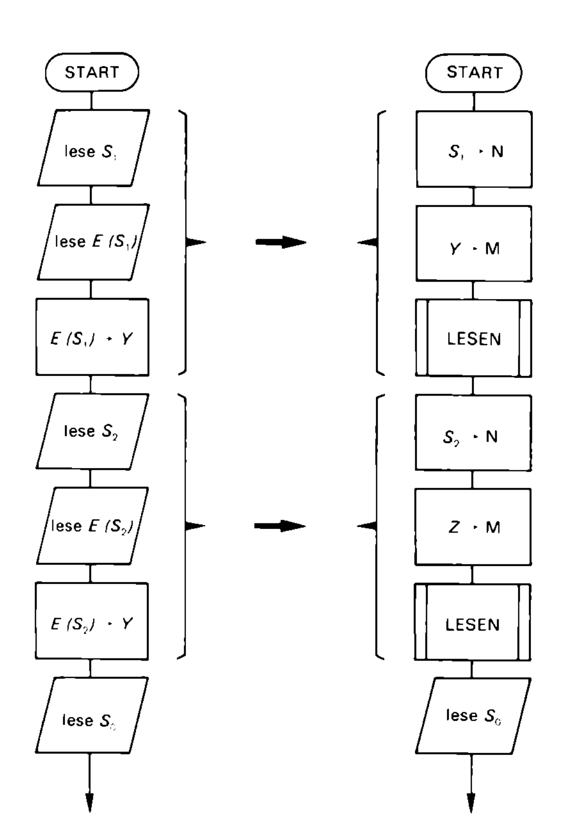


Bild 8 Modifiziertes Hauptprogramm mit speziellem Lese-Unterprogramm

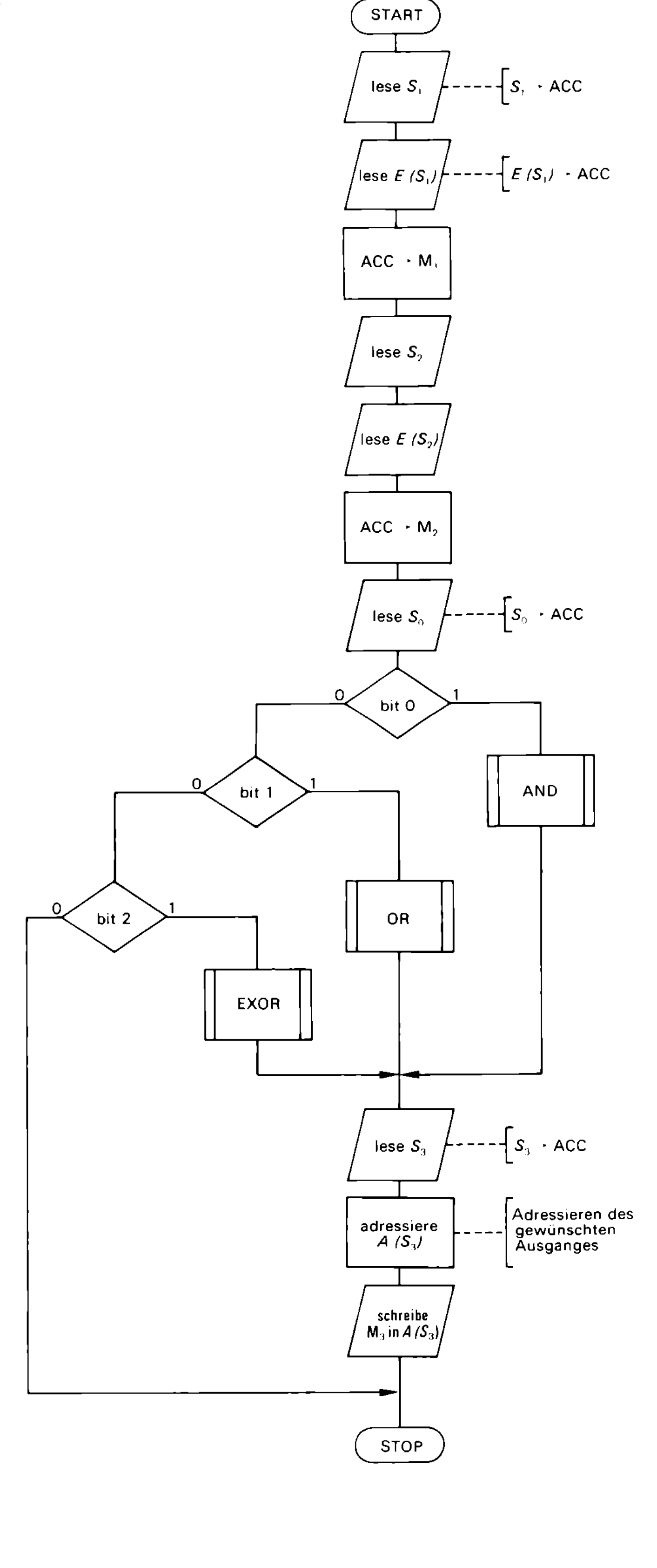


Bild 9 Detailliertes Hauptprogramm

Beim angeführten Beispiel lohnt sich dieser Zusatz aber schwerlich. Ob die Unterprogramme AND, OR und EXOR, wie oben angedeutet, nur aus einer einzigen Instruktion bestehen oder ob jedes Bit-Paar einzeln verarbeitet werden muss, hängt von der Komplexität des Instruktionssatzes des verwendeten Prozessors ab. Ebenso wird der Aufwand für die S_0-Entscheidung stark von den angebotenen Instruktionen abhängen.
Wir wollen nun das obige Flussdiagramm auf die Möglichkeiten eines zurzeit gängigen MPs umschreiben. Wir haben hier eine Anzahl Speicher (INX), den Akkumulator (ACC) sowie Eingänge (E, S) und Ausgänge (A) zur Verfügung. Jede dieser Einheiten enthält 4 bit. Es ist noch anzumerken, dass das Lesen bzw. Schreiben über den Akkumulator erfolgt. Somit sieht unser Flussdiagramm wie in Bild 9 aus.
Die entsprechenden Unterprogramme sind in Bild 10 dargestellt.

4.3 Beispiel Rechner

Ohne grosse Änderung können wir obiges Modell an ein populäres Produkt anpassen – den Taschenrechner.
Die Unterprogramme – bisher logische Operationen – werden durch arithmetische ersetzt, die Eingänge entsprechen den Eingaberegistern, und die Ausgänge reduzieren sich auf einen einzelnen, nämlich die Anzeige. Die Steuersignale kommen von den entsprechenden Funktionstasten. Auf diese Art einem 4-Spezies-Taschenrechner Konkurrenz zu machen, dürfte wirtschaftlich gesehen schwierig sein. Doch lässt sich leicht vorstellen, dass wir auf einfache Art und Weise unser Programm auf Probleme der höheren Mathematik ausbauen können. Es lassen sich auch Unterprogramme für ganze Problemkomplexe vorstellen, welche mit nur einem Tastendruck aufgerufen werden.
Vorerst wollen wir bei den Rechnern bleiben. Als weiteres Beispiel soll uns hier die Addition von mehrstelligen Dezimalzahlen dienen. In der bisherigen Technik würde für jede zusätzliche Stelle ein weiteres Addierwerk benötigt werden. Mit einem Prozessor kann dabei einiges gespart werden. Wir erhöhen für jede weitere Stelle nur die Anzahl der Durchläufe durch das addierende Unterprogramm, d.h. die einzige Änderung beruht auf dem Wechseln eines Speicherinhaltes.
Ein entsprechendes Flussdiagramm ist in Bild 11 gezeichnet.
Wir wollen diese Addition genauer durchspielen. Es sollen zwei Ziffern mit verschiedenen Stellenzahlen addiert werden. Bei der Ziffereingabe werden wir also die Ziffernzahl abspeichern. Nach der Eingabe haben wir folglich die nachstehenden Grössen im Speicher:

Speicher M_1 = Zahl A
Speicher M_2 = Zahl B
Speicher M_3 = Stellenzahl von A
Speicher M_4 = Stellenzahl von B

Das Resultat der Addition soll übrigens in M_1 gespeichert und von hier aus zur Anzeige gebracht werden.
Ungeachtet all dieser Informationen wollen wir zuerst ein Unterprogramm formulieren, welches zwei einzelne Ziffern mit Übertrag (Carry = CY) addieren kann:

$S = A + B + CY$

Wenn wir diese Addition im BCD-Code durchführen, dürfen wir eine gewisse Korrektur nicht vergessen. Die Summe von zwei

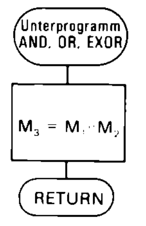


Bild 10 Unterprogramm zu Hauptprogramm nach Bild 9
[illegible] irgendwelche Operation

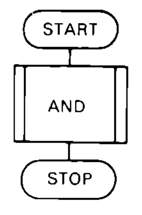


Bild 11 Flussdiagramm zur Addition zweier Zahlen

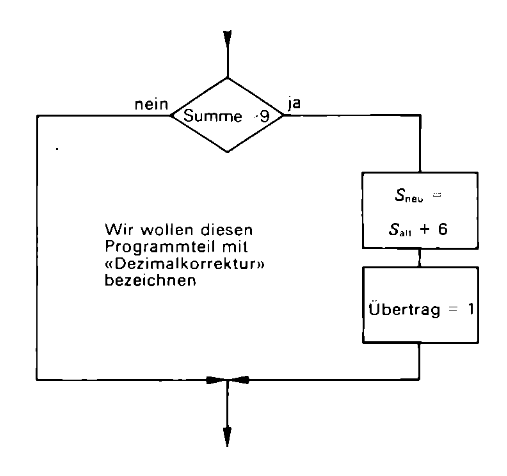


Bild 12 Flussdiagramm zur Dezimalkorrektur

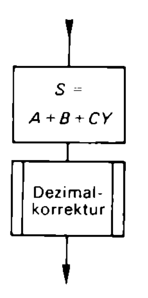


Bild 13 Programmteil zur korrekten Addition zweier BCD-Zahlen

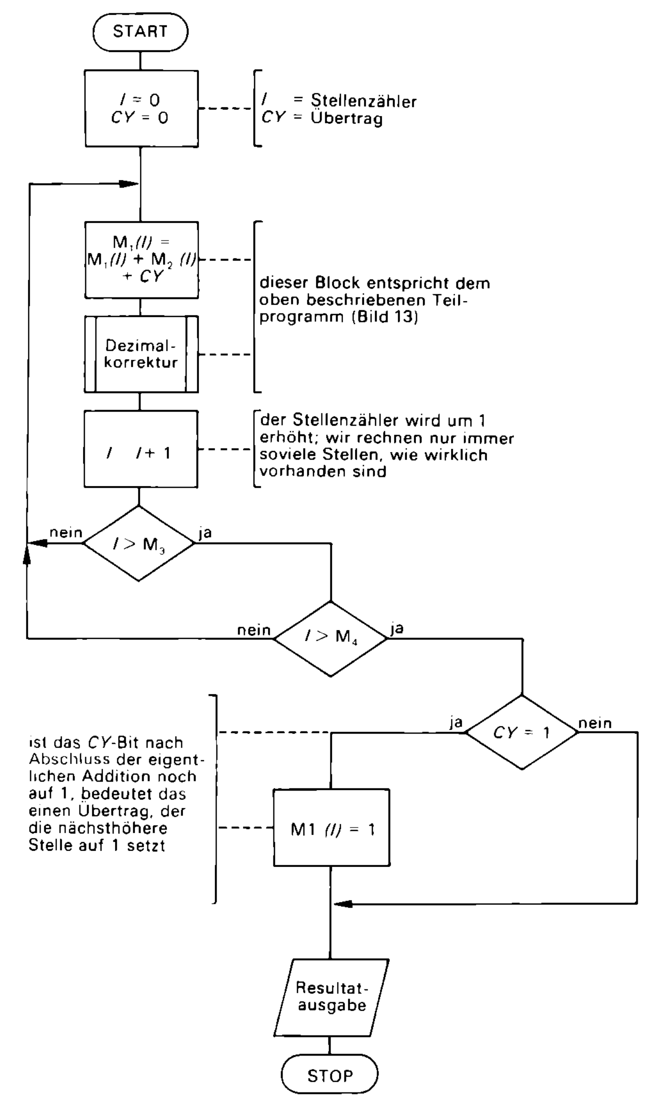


Bild 14 Programm zur Addition zweier mehrstelliger Zahlen

BCD-Zahlen kann bekanntlich grösser als 9 werden, liegt also nicht mehr im BCD-System.
Beispiel:

dezimal	binär	
5	0101	
+8	1000	
13	1101	→ diese Zahl liegt nicht mehr im BCD-System

Demzufolge ist eine Anpassung nötig. Ist die Summe grösser als 9, müssen 6 zur erreichten Summe addiert werden und der Übertrag (*CY*) wird auf 1 gesetzt (Bild 12).

5	0101	
+8	1000	
13	1101	→ grösser als 9
	+0110	→ Korrektur
	1 0011	
	1 3	→ dezimal

Das Unterprogramm für zwei Ziffern lautet nun wie Bild 13.
Die ganzen Zahlen haben aber mehrere Stellen. Das Hauptprogramm wird nun das Unterprogramm zur Zifferaddition so viele Male anlaufen, wie der höhere Stellenwert (M_3 oder M_4) der Eingabezahlen angibt (Bild 14).
Wenn natürlich der Zähler schon die maximale Speicher- bzw. Anzeigekapazität angibt, können wir auch keine höherwertige Stelle mehr ansteuern; somit bleibt uns nur noch, auf eine vorgegebene Art und Weise einen Überlauf anzudeuten. Auf analoge Weise wie die obige Addition lassen sich alle weiteren Funktionen eines Rechners programmieren.

5. Zusammenfassung

Die Anwendung eines Mikroprozessors erfordert ein Umlernen von der bisherigen Schaltungsentwicklung. Ein wichtiger Schritt auf diesem Weg ist die Erstellung eines Flussdiagrammes. Zuerst muss das zu lösende Problem mit all seinen Konsequenzen klar erkannt und definiert werden. Daraus lässt sich ein Flussdiagramm durch logische und zeitliche Aufgliederung gewinnen. Es darf dabei nie vergessen werden, dass ein Prozessor nur sequentielle Information verarbeiten kann.

Literatur:

MCS-4 Language Programming Manual, Intel Corp., Dec. 1973.
Microprocessors and their applications, ee/systems engineering today, Nov. 1973, Dec. 1973, Jan. 1974.
Introduction to Programming, Vol. 1, Digital Equipment Corp., Sept. 1970.

Digitale Halbleiterspeicher

Seit einigen Jahren werden in stetig zunehmender Zahl digitale integrierte Halbleiterspeicher angeboten. Ihr Einsatz bringt in vielen Anwendungen wesentliche Vorteile und gestattet andere Aufgaben erst wirkungsvoll zu lösen. In den ersten fünf Kapiteln dieses Aufsatzes werden die verschiedenen Speicherarten anhand leichtverständlicher analoger Beispiele eingeführt, die Organisationsformen, Schaltungsarten und Technologien soweit besprochen, wie es aus der Sicht des allgemein interessierten Anwenders notwendig erscheint, um die verschiedenen Einsatzmöglichkeiten zu erkennen. In den vier anschliessenden Kapiteln werden typische Speicherschaltungen im Detail besprochen, eine Reihe repräsentativer Produkte vorgestellt und zur Anregung eigener Ideen Anwendungsbeispiele aufgeführt. Dies alles mit dem Ziel, dem Leser ein einigermassen abgerundetes Bild über die Arten, Eigenschaften und Anwendungsmöglichkeiten moderner digitaler Halbleiterspeicher zu vermitteln, ohne aber einen Anspruch auf Vollständigkeit zu erheben.

1. Einleitung

In den letzten Jahren hat auf dem Gebiet der Digitaltechnik mit der allgemeinen Verfügbarkeit integrierter Halbleiterspeicher eine bedeutende Umwälzung eingesetzt. Hat noch bis zum Ende der sechziger Jahre der Magnetkernspeicher als rein elektronisches Speichermedium in allen Bereichen absolut dominiert, so liegt bereits heute der Wert der jährlich verkauften Halbleiterspeicher bei demjenigen der Kernspeicher. In Rechenanlagen werden diese zunehmend durch Halbleiterspeicher ersetzt, und Spekulationen gehen sogar so weit, dass ein Übergang von den elektromechanischen Magnetplattenspeichern («Disks») zu elektronischen integrierten Schieberegistern («Silicon Disks») vorausgesagt wird.

Kernspeicher und auch Magnetplatten- und Bandspeicher benötigen einen sehr grossen, von der Speicherkapazität weitgehend unabhängigen Aufwand an Hilfsschaltungen. Ihr Einsatz ist daher nur bei grösseren Speichervolumen wirtschaftlich möglich. Demgegenüber ist der Preis eines Halbleiterspeichers im wesentlichen nur von der Speichergrösse abhängig, so dass ein Einsatz bereits bei kleinem Bedarf an Speicherplätzen möglich wird. Damit eröffnet sich eine Vielzahl von *Anwendungsmöglichkeiten auch ausserhalb der Computertechnik*, wo bisher auf die Anwendung von Speichern grösstenteils überhaupt verzichtet werden musste. Zudem lassen sich viele Probleme der Digitaltechnik oft durch den Einsatz von Speichern auf überraschend einfache Weise lösen. Als Beispiele seien nur die Codeumwandlung und die Zeichenerzeugung mit Festwertspeichern erwähnt.

Da die integrierten Speicherschaltungen in zunehmendem Masse bereits alle zu ihrem Betrieb notwendigen Hilfsschaltungen, wie Mehrphasentaktgeneratoren, Decodier- und Steuerschaltungen, enthalten und zudem direkt durch TTL-Signale angesteuert werden können oder selbst TTL-kompatible Signale abgeben, bietet ihre Anwendung kaum mehr Probleme als diejenige irgendwelcher logischer Schaltungen. Weiter können die neuen N-Kanal-MOS- und die komplementären MOS-Schaltungen mit nur einer 5-V-Speisespannung betrieben werden, so dass eigentlich nichts mehr, ausser vielleicht mangelnde Kenntnisse und Informationen über die Halbleiterspeicher, ihrer breiten Anwendung entgegenstehen sollte. Das Ziel der folgenden Ausführungen ist es, diese Lücken zu füllen und damit dem allgemein interessierten Anwender, nicht dem Spezialisten, die Halbleiterspeichertechnik näherzubringen.

2. Speicherarten, Grundbegriffe

Bevor wir uns eingehend mit den verschiedenen Halbleiterspeichern, ihren Schaltungen, Technologien und Anwendungen befassen, sollen die einzelnen Typen charakterisiert und einige Grundbegriffe eingeführt werden.

2.1 RAM

Holen wir einmal einen Freund vom Flugplatz ab, der uns einen Koffer voll der neuesten Speicher-ICs aus Amerika mitbringt. Dort angekommen, müssen wir zuerst unser Fahrzeug auf dem Parkplatz abstellen (= speichern). Dabei können wir direkt ein beliebiges noch freies Parkfeld (= Speicherplatz, Speicherzelle) wählen, ohne den Inhalt der anderen Speicherplätze zu verändern oder zu verschieben. Um den Wagen wiederzufinden, brauchen wir uns nur die Nummer (= Adresse) des Parkfeldes zu merken. Wir haben in Form des Parkplatzes einen *Speicher mit direktem oder wahlfreiem Zugriff*, ein *random access memory* oder kurz *RAM*, gefunden. Die Speicherplätze sind darin zweidimensional in einer Ebene angeordnet. Jeder beliebige Speicherplatz kann einzeln und direkt über seine Adresse erreicht werden. Die Zeit, welche dazu benötigt wird, ist für jeden Platz gleich und wird als *Zugriffszeit* oder *access time* bezeichnet. In dieser Hinsicht scheint das Beispiel des Parkplatzes schlecht gewählt zu sein, wo es doch offensichtlich länger dauert, das hinterste Parkfeld zu erreichen als eines der vordersten. Aber mit dem Erreichen des Parkfeldes ist das Parkieren ja noch lange nicht abgeschlossen. Zuerst muss der Wagen richtig stehen, die Insassen müssen sich abschnallen, die Fenster schliessen, die Damen ihre Haartracht ordnen, Taschen -

und Mappen zusammensuchen, aussteigen und abschliessen. Dass demgegenüber die reine Fahrzeit vernachlässigt werden kann, steht wohl ausser Zweifel. Ganz ähnlich liegen die Verhältnisse beim RAM-IC. Hier setzt sich die Zugriffszeit aus den für jeden Speicherplatz gleichen Schaltzeiten der Steuer- und Treiberlogik, dem *circuit delay*, und der eigentlichen Ausbreitungsgeschwindigkeit der Information in der Speicherebene, dem *array delay*, zusammen (array = Mehrfachanordnung). Da alle Speicherzellen eng benachbart auf einem winzigen Siliziumplättchen von wenigen Millimetern Kantenlänge untergebracht sind, tritt die letztere gegenüber der ersteren kaum in Erscheinung. Nach [1] weist zum Beispiel ein typischer TTL-Speicher eine Lesezugriffszeit von 70 ns auf, wovon höchstens 10 ns auf den array delay entfallen.

2.2 ROM, PROM, RePROM

Nun jedoch zurück zum Flugplatz. An einer Informationstafel wollen wir uns über die genaue Ankunftszeit unseres Freundes orientieren. Wir suchen also darauf den Speicherplatz mit der Adresse «Flug 718 von New York» und finden dort die Ankunftszeit «9.45» gespeichert. Auf die gleiche Weise können wir die Ankunftszeit jedes beliebigen Fluges direkt herauslesen, es ist uns jedoch nicht möglich, eine der eingeschriebenen Informationen zu verändern. Die Informationstafel stellt für uns einen *Festwert-* oder *Nur-Lese-Speicher*, ein *read only memory* oder kurz *ROM* dar. Sein Aufbau ist grundsätzlich gleich wie der des RAMs, mit dem entscheidenden Unterschied, dass die Informationen bei der Herstellung unveränderbar in die einzelnen Speicherzellen programmiert werden. Dies geschieht normalerweise durch die Verwendung einer speziellen Metallisierungsmaske. Wie beim RAM kann jedoch die Information jeder einzelnen Speicherzelle direkt über ihre Adresse in der gleich kurzen Zugriffszeit herausgelesen werden. Die Programmierung eines ROMs ist wegen der notwendigen speziellen Maske eine aufwendige Sache und kommt nur in Frage, wenn eine grössere Anzahl gleich programmierter Speicher benötigt wird. Diesen Nachteil vermeidet der *programmierbare Festwertspeicher*, das *programmable read only memory* oder *PROM*. Bei ihm kann die Information einmal vom Anwender mit einem Programmiergerät nicht wieder löschbar in den Speicher eingeschrieben werden. Entsprechende Geräte sind im Handel erhältlich, sie sind jedoch oft nur für ein PROM-Modell brauchbar. Bild 1a zeigt ein am Institut für Fernmeldetechnik der ETH entwickeltes Programmiergerät für das Signetics-PROM 8223, die Bilder 1b und 1c zeigen Beispiele kommerzieller Geräte. Eine dritte Version, der *löschbare Festwertspeicher*, das *reprogrammable read only memory* oder *RePROM*, schliesslich gestattet, die einmal eingeschriebene Information wieder zu löschen und den Speicher wiederholt neu zu programmieren. Das RePROM kann jedoch nicht als Ersatz eines RAMs angesehen werden, da das Löschen und Neuprogrammieren nur sehr langsam erfolgen kann und wieder oft mit einem Spezialgerät vorgenommen werden muss (einige RePROMs lassen sich mit einer Quarzlampe löschen). Zudem ist es auch meistens nicht möglich, nur einzelne Speicherzellen zu löschen; es wird fast immer der gesamte Speicherinhalt vernichtet (siehe auch unter 5.2). Einen für gewisse Anwendungen entscheidenden Vorteil gegenüber allen anderen Halbleiterspeichern haben jedoch alle drei ROM-Typen, ihr einmal programmierter Inhalt geht auch beim Wegfall der Speisespannung nicht verloren, es sind *nichtflüchtige Speicher* oder *non volatile memories*.

2.3 SR

Inzwischen haben wir erfahren, dass ein Teil des Flugplatzpersonals streikt, so dass die angekommenen Passagiere ihr Gepäck selbst befördern müssen. Dabei haben sich alle in einer langen Reihe, eine Transportkette bildend, zwischen der Zollabfertigung und den Gepäckwagen aufgestellt. Dort wird nun ein Gepäckstück nach dem andern aufgenommen und durch die Transportkette auf die Reise geschickt, wo alle im Takt von Person zu Person bis ans Ende weitergegeben werden. Da dort gelegentlich Schwierigkeiten mit der Abnahme bestehen, muss der Transportvorgang zeitweise unterbrochen werden, wobei die Gepäckstücke während dieser Zeit beim jeweiligen Glied der Kette gespeichert werden. Die analoge elektronische Speicherschaltung zur Transportkette ist das *Schieberegister*, *shift register* oder kurz *SR*. Es ist ein Speicher vom Typ *first in first out*, abgekürzt *FIFO*, bei welchem die Information nur in der Reihenfolge ihrer Eingabe wieder ausgelesen werden kann (Silospeicher). Was zuerst eingegeben wurde, erscheint auch zuerst wieder am Ausgang, wenn es den ganzen Speicher durchlaufen hat. Die Speicherzellen sind eindimensional in einer Linie angeordnet, und jede kann ihre Information nur an die nächstfolgende weitergeben bzw. von der vorhergehenden empfangen. Dabei erfolgt die Verschiebung gleichzeitig im ganzen Register um jeweils einen Speicherplatz pro Taktschritt. Im Gegensatz zum RAM und ROM ist es beim Schieberegister nicht möglich, eine beliebige Information direkt herauszuholen, vielmehr muss diese zuerst über eine je nach ihrer momentanen Lage verschiedene Anzahl von Speicherzellen bis zum Ausgang weitergegeben werden. Die Zugriffszeit ist demnach nicht mehr konstant und kann bei langen Schieberegistern recht grosse Werte erreichen. Um zu verhindern, dass die vor der gesuchten Information liegenden Speicherinhalte verlorengehen, wird, wenn das Schieberegister, einmal gefüllt ist, der Ausgang mit dem Eingang verbunden, wodurch ein geschlossener Kreis, ein *recirculating shift register* entsteht. Die dazu notwendige Umschaltlogik ist bei vielen Typen bereits eingebaut. Bei

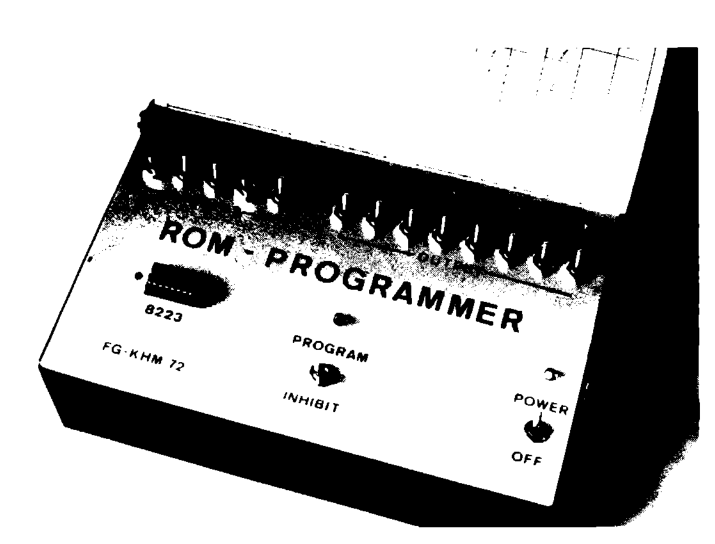


Bild 1a Ansicht eines einfachen Programmiergerätes für das Signetics-PROM 8223

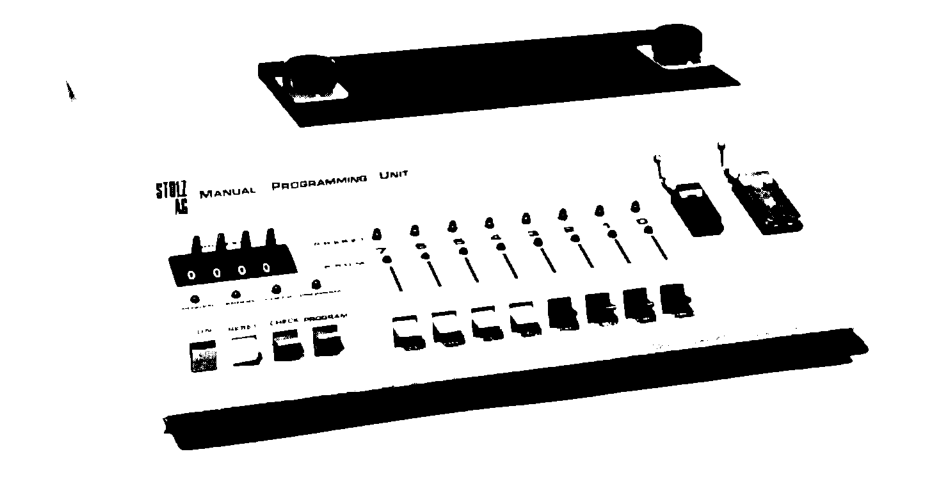


Bild 1b Kommerzielles Handprogrammiergerät für verschiedene PROM-Typen
(Stolz AG, CH-8968 Mutschellen)



Bild 1c Kommerzielles automatisches Programmiergerät für verschiedene PROM-Typen
(Stolz AG, CH-8968 Mutschellen)

dynamischen Schieberegistern ist das Umlaufen der Information sogar dauernd notwendig, da die einzelnen Speicherzellen diese in Form der Ladung auf einem Kondensator nur für eine kurze Zeit, maximal einige Millisekunden, speichern können. Bei der Weitergabe an die nächste Speicherzelle findet jeweils eine Erneuerung oder Auffrischung der Information statt. Um trotz dieser dauernden Verschiebung die Orientierung über die Lage der einzelnen Informationen nicht zu verlieren, müssen die Taktschritte in einem *Adressenzähler* gezählt werden. Dessen Stand gibt dann jeweils an, welche Information sich gerade am Registerausgang befindet und somit ausgelesen werden kann. Auch ein RAM kann als dynamischer Speicher ausgeführt werden. Dort müssen im Abstand von ebenfalls höchstens einigen Millisekunden spezielle *Auffrisch-* oder *refresh-Zyklen* eingeschaltet werden, damit die Information nicht verlorengeht. Im Gegensatz dazu ist ein ROM immer statisch. Die Vorteile der dynamischen Speicher gegenüber den statischen, deren Zellen eine Information beliebig lange halten können (beim RAM und SR unter der Bedingung, dass die Speisespannung nicht wegfällt), liegen in einem einfacheren Aufbau der Speicherzellen, bedeutend geringerer Verlustleistung und bei Schieberegistern in einer höheren Geschwindigkeit.

2.4 Sondertypen

Zur Vervollständigung werden nun noch einige Speichertypen erwähnt, bei deren Einsatz das Hauptgewicht weniger auf der reinen Speicherung als auf den damit durchführbaren Operationen liegt. Beim *first in last out*, abgekürzt *FILO*-Speicher, können die Informationen nur in der umgekehrten Reihenfolge ihrer Eingabe wieder herausgelesen werden. Seine Funktionsweise kann anschaulich am Beispiel einer in eine Sackgasse marschierenden Marschkolonne illustriert werden: Spätestens wenn der vorderste Mann das Ende der Sackgasse erreicht hat (der Speicher voll ist), erfolgt der Befehl «rechtsumkehrt», und die Kolonne verlässt diese in der umgekehrten Reihenfolge mit dem vorherigen Schlussmann an der Spitze wieder. Realisiert wird dieser Speicher mit einem links-rechts Schieberegister, bei welchem die Information vorwärts und rückwärts übertragen werden kann. Ein Anwendungsbeispiel sind

die vier Rechenregister der hp-Taschenrechner. Anstelle von FILO verwendet man auch den Ausdruck *stack*, was auf deutsch etwa als Stapelspeicher im Gegensatz zum Silospeicher (FIFO) übersetzt werden kann.
Der *assoziativ* oder *inhaltadressierbare Speicher*, das *content adressable memory*, kurz *CAM*, gestattet, den Speicherinhalt direkt mit einer gegebenen Information zu vergleichen. Dadurch kann festgestellt werden, ob sich eine bestimmte Information schon im Speicher befindet, oder es kann nach Informationen bestimmter Charakteristiken gesucht werden – in einem Adressenspeicher zum Beispiel nach den Bewohnern einer bestimmten Strasse.
Das *multiport memory* schliesslich besitzt mehrere Ein- Ausgänge, so dass gleichzeitig verschiedene Informationen herausgelesen oder gelesen und eingeschrieben werden können.
In neuerer Zeit verwendet man auch den Ausdruck CROM. Diesen Ausdruck findet man in der Mikrocomputertechnik, wenn ein ROM als Steuereinheit (control) eingesetzt wird.

Bild 2 Speicherung eines 4-bit-Wortes in vier RAM-Speicherebenen

3. Speicherorganisation

Bis jetzt wurde immer nur von der Speicherung von Informationen gesprochen, ohne diesen Begriff näher zu umschreiben und auf die Speicherorganisation einzugehen. In den einzelnen Speicherzellen können nur die beiden Zeichen 0 oder 1 in Form einer hohen oder tiefen Spannung gespeichert werden. Nach der Informationstheorie entspricht dies einem Informationsgehalt von einem *Bit* (Zeichen: bit) oder einer *Binärziffer*. Gehen wir zurück zum ersten Beispiel des Parkplatzes, so kann zum Beispiel für jedes Parkfeld in einer zugeordneten Speicherzelle eines RAMs die Information «frei» oder «besetzt» eingetragen werden. Soll jedoch zur Überwachung der Parkzeit die Ankunftszeit der Fahrzeuge gespeichert werden, so muss diese zuerst in eine *Folge von Binärziffern*, zum Beispiel im BCD-Code codiert, umgewandelt werden. Die Information liegt dann in Form eines *binären Wortes* aus mehreren Binärziffern vor. Dieses benötigt nun zur Speicherung mehrere Speicherzellen, für jedes Bit eine.
Bei der normalerweise angewandten *parallelen Speicherung* wird jedes Bit eines Wortes unter der gleichen Adresse in einer separaten Speicherebene untergebracht. Bild 2 zeigt wie zum Beispiel die vier Bits einer BCD-codierten Dezimalzahl in vier RAM-Speicherebenen als 4-bit-Wort unter der Adresse 32 abgespeichert werden. Dabei gibt die erste Adressenziffer die Nummer der Spalte (englisch column), die zweite diejenige der Zeile (englisch row) der Speicherebene an, in welcher sich der Speicherplatz befindet. Diese Art der Adressierung, jedoch binär codiert, ist allgemein gebräuchlich und wird bei der Behandlung des ROMs nochmals besprochen. Der grösste Teil der erhältlichen RAM-ICs ist *bit-organisiert* und enthält jeweils nur eine Speicherebene. Darin kann pro Adresse nur ein Bit oder ein zu einem Bit degeneriertes Wort gespeichert werden. Einige RAM-Typen und alle ROMs sind dagegen *wortorganisiert* und enthalten in einer Schaltung mehrere Speicherebenen (welche nicht unbedingt örtlich voneinander getrennt sein müssen!), so dass pro Adresse ein ganzes Wort gespeichert werden kann.
Zur Charakterisierung eines Speichers wird die Anzahl der Speicherplätze, die *Kapazität des Speichers*, von zum Beispiel 256 bit (fast ausnahmslos eine ganzzahlige Potenz von 2) und die *Organisation* in zum Beispiel 32 Worten zu 8 bit beim schon erwähnten Signetics-Rom 8223 angegeben. Bei diesem Speicher kann demnach unter jeder der 32 Adressen je ein 8-bit-Wort ausgelesen werden. Die Speicherkapazität ist immer gleich dem Produkt aus der Anzahl der Worte und der Wortlänge. Häufig wird auch nur dieses angegeben, für das obige Beispiel also 32 × 8 bit.
Die *serielle Speicherung*, bei welcher die verschiedenen Bits eines Wortes in nebeneinander liegenden Speicherzellen eines einzigen Speichers untergebracht werden, wird weniger häufig und dann vorwiegend bei Schieberegistern angewendet. Da dabei alle Bits einzeln nacheinander eingeschrieben oder ausgelesen werden müssen, wird die minimale Zugriffszeit bei der seriellen Speicherung immer mindestens *n* mal grösser als bei der parallelen, wobei *n* die Wortlänge ist. Wenn jedoch die Verarbeitung der einzelnen Bits eines Wortes ebenfalls nacheinander erfolgt, wie in fast allen elektronischen Taschenrechnern, dann empfiehlt sich die serielle Speicherung mit einem Schieberegister.

4. Schaltungsarten

Wie alle integrierten Digitalschaltungen werden auch die Speicher in mehreren Schaltungsarten hergestellt. Es sind dies die als Logikfamilie weit verbreitete *TTL (Transistor-Transistor-Logik)*, *ECL (Emitter Coupled Logic)* und die *CMOS*- oder *COSMOS (Complementary Metal Oxide Semiconductor)*-Technik, währenddem die wichtigste Speicherschaltungsart, die *MOS (Metal Oxide Semiconductor)*-Technik als Logikfamilie praktisch nicht existiert. Die beiden ersten, TTL und ECL, arbeiten mit bipolaren NPN-Transistoren, in der MOS-Technik werden entweder P- oder N-Kanal-Feldeffekttransistoren verwendet, währenddem die letzte, CMOS, mit N- und P-Kanal-FETs aufgebaut wird. Um bei der Besprechung der Speicherschaltungstechnik nicht mehr auf die Eigenschaften der verschiedenen Schaltungsarten eingehen zu müssen, werden diese anschliessend kurz charakterisiert. In der Tabelle 1 sind einige typische Daten und Eigenschaften der verschiedenen Speicherschaltungsarten einander gegenübergestellt.

4.1 TTL

Die TTL-Technik ist heute noch unbestritten die Standardschaltungsart der Digitaltechnik. Speicherschaltungen (Festwertspeicher ausgenommen) werden jedoch nur in geringer Auswahl angeboten und nur dann eingesetzt, wenn die damit erreichbaren höhe-

ren Arbeitsgeschwindigkeiten unbedingt benötigt werden. Die Gründe dafür sind der hohe Leistungsverbrauch und die relativ komplizierte Technologie, welche verbunden mit einem grossen Flächenbedarf der Schaltungen zu hohen Preisen führen. Spezialitäten dieser Schaltungsart sind jedoch die im Abschnitt 2.4 aufgeführten Sondertypen Assoziativ- und Multiportspeicher. Längere Schieberegister werden dagegen überhaupt nicht hergestellt. Durch den Übergang von der normalen TTL-Bauweise mit gesättigten Transistoren zur *Schottky-Technik*, bei welcher durch eine nichtlineare Gegenkopplung der Transistoren mit Schottky-Dioden die Sättigung der Transistoren verhindert wird, kann entweder die Arbeitsgeschwindigkeit weiter erhöht oder die Verlustleistung gesenkt werden. Dynamische Schaltungen werden in TTL-Technik nicht hergestellt, da wegen der niederohmigen Steuerkreise viel zu grosse Kapazitäten realisiert werden müssten.

4.2 ECL

Die ECL-Technik wird ausschliesslich in schnellsten Logiksystemen angewendet. Sie arbeitet mit ungesättigten Bipolartransistoren nach dem Prinzip der Stromumschaltung durch übersteuerte Differenzverstärker. Die wenigen erhältlichen ECL-RAMs werden nur dort eingesetzt, wo extrem kurze Zugriffszeiten gefordert werden. Dies sind vor allem die temporären Speicher in den Rechenwerken schnellster Grossrechner.

4.3 P-MOS

Die P-MOS-Technik mit selbstsperrenden (enhancement) P-Kanal-Feldeffekttransistoren ist die älteste und heute noch am meisten verbreitete Speicherschaltungsart. Durch die Einführung neuer Herstellungsverfahren zur Verkleinerung der Transistorschwellenspannungen konnte der anfängliche Nachteil hoher notwendiger Speisespannungen und Signalamplituden behoben werden. Alle neueren P-MOS-Speicherschaltungen haben TTL-kompatible Ein- und Ausgänge. Im allgemeinen werden jedoch noch zwei, gelegentlich auch drei Speisespannungen benötigt. Durch die Anwendung dynamischer Schaltungstechniken unter Ausnutzung der ohnehin vorhandenen Gatekapazitäten der Transistoren als Kurzzeitspeicher ist es möglich, mit kleinen Verlustleistungen auszukommen.

4.4 N-MOS

Die neueste Schaltungsart der integrierten Halbleiterspeicher ist die N-MOS-Technik mit selbstsperrenden N-Kanal-Feldeffekttransistoren. Es wird erwartet, dass sie sich in kurzer Zeit als Standardschaltungsart durchsetzen wird. Gegenüber den P-MOS-Schaltungen bestehen zwei wesentliche Vorteile. Dank der rund dreimal höheren Trägerbeweglichkeit im N-Kanal ist es möglich, bei gleichen Abmessungen höhere Arbeitsgeschwindigkeiten oder bei gleicher Geschwindigkeit kleinere Abmessungen der Schaltungen zu erreichen. Der Betrieb mit einer 5-V-Speisespannung ist möglich. Dadurch wird automatisch auch die Verlustleistung geringer, welche zusätzlich noch durch die dynamische Schaltungstechnik reduziert werden kann.

4.5 CMOS

Bei der CMOS-Technik werden selbstsperrende P- und N-Kanal-Feldeffekttransistoren im Gegentaktbetrieb verwendet. Dadurch ist es möglich, im Ruhezustand ohne Verlustleistung auszukommen. Wegen der notwendigen Isolationszonen zur Trennung der zwei komplementären Transistortypen benötigen die Schaltungen relativ viel Platz. Dies ist mit der komplizierteren Technologie ein wesentlicher Grund für die noch hohen Preise. Die Störsicherheit ist ausgezeichnet, und die Schaltungen können über einen grossen Speisespannungsbereich betrieben werden, speziell im Zusammenwirken mit TTL-Bausteinen an + 5 V.

5. Technologien

Die Kenntnis der wichtigsten Eigenschaften der für die Herstellung integrierter Halbleiterspeicher angewandten Technologien ist unumgänglich, um die Unterschiede zwischen den verschiedenen Varianten zu verstehen. Hauptsächlich bei den MOS-Speichern existiert eine Vielzahl verschiedener Herstellungsverfahren, und neue kommen ständig hinzu. Die wichtigsten werden in diesem Kapitel zusammengestellt, und ihr Einfluss auf die Eigenschaften der damit realisierten Speicherschaltungen wird angegeben.

5.1 Bipolare Technologien

Obschon zur Herstellung bipolarer integrierter Schaltungen verschiedene Varianten der *Planar-epitaxial-Technologie* bestehen, können keine wesentlichen Unterschiede in den Eigenschaften der damit hergestellten integrierten Digitalschaltungen festgestellt werden. Es ist daher aus der Sicht des Anwenders nicht notwendig, näher darauf einzugehen. In der Fabrikation sind die bipolaren Schaltungen mit etwa 120 Prozessschritten eindeutig am kompliziertesten [2]. Da die Vorgänge jedoch dank der grossen Erfahrung und den nicht allzu hohen Anforderungen an die Konstanz der einzelnen Parameter sehr gut beherrscht werden, lassen sich trotzdem hohe Ausbeuten und damit günstige Preise erreichen.

5.2 MOS-Technologien

Bei allen drei Schaltungsarten mit MOS-Feldeffekttransistoren ist die Tiefhaltung und Konstanz der Schwellenspannungen, die Gate-Source-Spannung, bei welcher der Übergang vom leitenden in den gesperrten Zustand erfolgt, das schwierigste Herstellungsproblem. Niedrige Schwellenspannungen (in der Grösse von 2 bis 3 V) sind jedoch Voraussetzung für die direkte Kombinierbarkeit mit TTL-Schaltungen und den Betrieb mit 5-V-Speisung. Ein fast unentbehrliches Hilfsmittel dazu ist die *Ionenimplantation*. Damit ist es möglich, mit der üblichen Diffusionstechnik nicht realisierbare kritische Dotierungen mit hoher elektrischer und geometrischer Genauigkeit herzustellen. Die Störatome werden dabei mit hoher Geschwindigkeit in die Halbleiterplättchen eingeschossen. Die Anzahl der zur Herstellung notwendigen Prozessschritte ist bei der N- und P-MOS-Technologie mit 25 weitaus am geringsten [2]. Für CMOS-Schaltungen ist bereits die doppelte Anzahl notwendig. Im Gegensatz zur Bipolartechnologie sind jedoch die Anforderungen an die einzelnen Prozessschritte bedeutend höher, so dass der fabrikationstechnische Vorteil der MOS-Technologien nicht so sehr ins Gewicht fällt.

Bei der häufig angewandten *Silicon-Gate*-Bauweise wird die normalerweise aus Aluminium bestehende Gate-Elektrode durch polykristallines Silizium ersetzt. Damit wird die Schwellenspannung um $\approx$ 1 V reduziert, und dank der höheren Temperaturfestigkeit ist es möglich, die Drain-und-Surce-Diffusion erst nach der Gate-

Herstellung vorzunehmen, wodurch die Gate-Elektrode gerade als Maske zur Abschirmung des Kanals verwendet werden kann (*Self-aligning-gate*-Technik). Dadurch können kleinere Abmessungen und grössere Schaltgeschwindigkeiten erreicht werden, da die Ungenauigkeiten einer zusätzlichen Maske und Überlappungen der Gate- und Drain-Source-Bereiche vermieden werden.
Durch die Einführung einer zusätzlichen Silizium-Nitrid-Schicht zwischen der Gate-Elektrode und dem Gate-Oxid ist ebenfalls eine Reduktion der Schwellenspannung möglich. Zudem können mit der Folge: *Metall-*, *Nitrid-*, *Oxid-*, *Silizium-*, kurz der *MNOS*-Technik, elektrisch programmier- und löschbare Festwertspeicher realisiert werden. Durch das Anlegen einer hohen Gate-Spannung ($\approx$ 35 V) wird die Siliziumnitrid-Siliziumoxid-Grenzschicht über einen Tunneleffektmechanismus aufgeladen, wodurch die Schwellenspannung des Transistors verändert wird. Dabei bleibt die auf diese Weise programmierte hohe oder tiefe Schwellenspannung als Information über längere Zeit (Grössenordnung) Jahre erhalten [3].
Kommerzielle Produkte, die diese Technologie verwenden, sind oft auch unter dem Namen EAROM (electrically alterable ROM = elektrisch veränderbares ROM) bekannt. Das EAROM ist eine Art PROM, dessen Inhalt selektiv elektrisch gelöscht und geschrieben werden kann. Es braucht aber spezielle Signale, d.h. nicht die normalen Schreibsignale, wie sie in einem RAM verwendet werden. Das EAROM ist ein spezielles PROM und ist nicht mit einem normalen RAM zu verwechseln.
Mit der in naher Zukunft in der Produktion erwarteten *Silicon-on-Saphir-*, kurz *SOS*-Technologie lassen sich Schaltungen mit etwa zehnmal höherer Arbeitsgeschwindigkeit herstellen. Dabei wird anstelle von Silizium als Substrat der Isolator Saphir (oder auch Spinel) zum Aufbau der Schaltungen verwendet, wodurch die parasitären Kapazitäten drastisch verkleinert und die einzelnen Transistoren praktisch ideal gegeneinander isoliert werden. Dadurch entfallen bei der CMOS-Schaltungsart die sonst viel Platz beanspruchenden speziellen Isolationszonen.

Tabelle 1 Gegenüberstellung einiger typischer Daten statischer RAMs in den verschiedenen Schaltungsarten

	TTL	ECL	P-MOS	N-MOS	CMOS
Zugriffszeit in ns	50	10	750	250	250
Ruheleistung pro bit in mW	3	4	1	0,1	< 1 μW
Preis pro bit in sFr.	0.25	0.50	0.03	0.06	0.30
TTL-kompatibel	ja	nein	ja	ja	ja
Speisespannung in V	+ 5	− 5,2	+ 5 − 12	+ 5	+ 4 ... + 15

Literaturverzeichnis

[1] D. A. Hoges, «Large-Capacity Semiconductor Memory», Proc. IEEE, Vol. 56, No. 7, July 1968, p. 1155, Tabelle III.

[2] P. R. Casse, «Technology of Recent FET Microelectronic Circuits», Scientia Electrica, Vol. 17, No. 3, 1971, p. 78, Tabelle III.

[3] J. H. Gilder, «MNOS memory upstaging MOS and fixed heads in some areas», Electronic Design, Vol. 21, No. 18, September 1973, p. 28.

6. ROM, PROM, RePROM

In Kapitel 2 und 3 haben wir gesehen, dass bei ROM-Typen und beim RAM die Speicherzellen zweidimensional in einer Ebene angeordnet sind. Bei der Realisierung dieser Speicher gibt es grundsätzlich zwei verschiedene Probleme zu lösen: Es müssen Speicherzellen gefunden werden, welche die gewünschten Informationen speichern können, und jede einzelne Speicherzelle muss mit dem Eingang und Ausgang des Speichers verbunden werden können. Am Beispiel des bereits zitierten Parkplatzes illustriert: Es müssen sowohl die einzelnen Parkfelder vorhanden sein als auch die notwendigen Verbindungswege zur Ein- und Ausfahrt. Im Falle des «read only memory» sind beide Probleme einfach zu lösen. Da der Speicher bei der Herstellung bereits endgültig vorprogrammiert wird, können die Speicherzellen aus einem einzigen Element bestehen, welches entsprechend der darzustellenden Information 0 oder 1 entweder funktionstüchtig oder defekt hergestellt wird.

6.1 Aufbau

Da ein ROM immer wortorganisiert ist, ist die Herstellung der Verbindung zwischen den Speicherzellen und den Ausgangsanschlüssen relativ einfach, da mit jeder Adresse immer eine ganze Speicherzellengruppe, deren Information ein Wort bildet, ausgewählt wird. In Bild 3 ist der grundsätzliche Aufbau eines ROM dargestellt. Die Speicherebene wird durch p horizontale Wortleitungen und m vertikale Bitleitungen unterteilt. In jedem der $m \cdot p$ Kreuzungspunkte verbindet eine Diode als Speicherzelle die sich überschneidenden Leitungen. Man erkennt in diesem Bild die eigentlich längst bekannte Diodenmatrix. Entsprechend den zu speichernden Informationen 1 oder 0 ist die Diode funktionstüchtig oder nur einseitig angeschlossen (d. h. eigentlich nicht vorhanden). Die Bitleitungen sind über Widerstände mit Masse verbunden und führen über Zwischenstufen zu den Ausgangsanschlüssen des Speichers. Die Wortleitungen sind an den Ausgängen eines Decoders angeschlossen, über dessen Eingänge die Adresse des gewünschten Wortes eingegeben wird. Mit den n Adressenbits können $2^n = p$ verschiedene Adressen gebildet werden. Durch den Decoder wird jeweils derjenige Ausgang (und damit die angeschlossene Wortleitung) auf das einer logischen 1 entsprechende positive Potential gehoben, dessen Nummer, binär codiert, identisch mit der eingestellten Adresse ist. Alle anderen Decoderausgänge und Wortleitungen sind auf Massepotential. Dadurch werden alle Bitleitungen, welche mit der gewählten Wortleitung durch eine auf 1 programmierte Speicherzelle, also eine funktionsfähige Diode, verbunden sind, über diese ebenfalls potentialmässig «hochgezogen». Alle andern Bitleitungen bleiben auf Massepotential. In Bild 3 ist als Beispiel die Adresse 010. . . .0 eingetragen. Diese entspricht der binär codierten Dezimalzahl 2. Folglich befindet sich der Ausgang X_2 des Decoders auf 1-Potential, und am Ausgang des Speichers erscheint das durch die an der Wortleitung 2 angeschlossenen Speicherzellen programmierte Ausgangssignal 1100. . . 1. Über den Anschluss CS, «Chip select» (häufig auch mit CE, «Chip enable», bezeichnet), kann der Speicher blockiert werden. Bei sogenannten «Tri state»-Ausgängen werden die Ausgangsstufen in einen hochohmigen Zustand gebracht, bei «Open collector»-Ausgängen werden die Ausgangstransistoren im Sperrzustand blockiert. Dadurch ist es möglich, mehrere Speicher parallel zu schalten und den jeweils gewünschten Speicher auszuwählen.

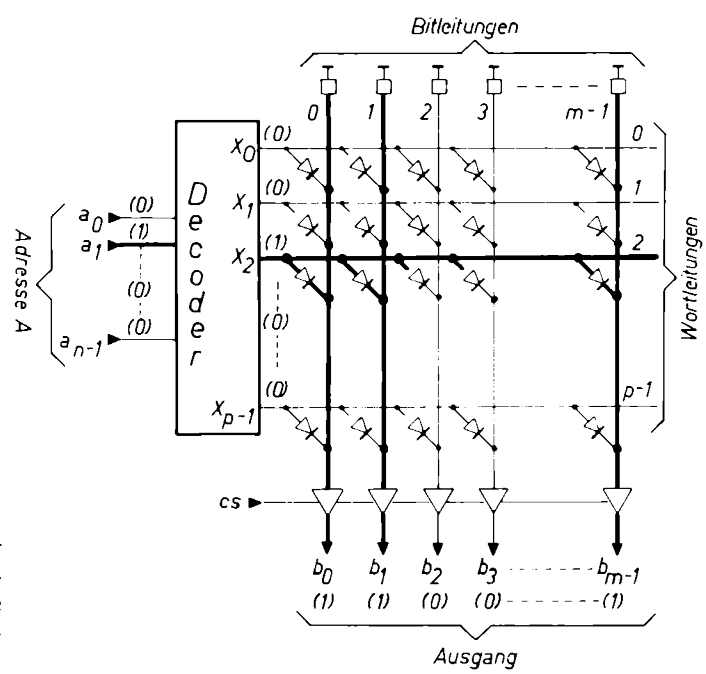


Bild 3 Aufbau eines wortorganisierten Festwertspeichers (ROM) mit Diodenspeicherzellen (Decoder mit aktivem Ausgang hoch)

6.2 Speicherzellen

Bei bipolaren ROMs in TTL-Technik werden anstatt Dioden (Bild 3) sehr oft auch Transistoren als Speicherzellen eingesetzt. Bild 4a zeigt eine entsprechende Schaltung. Die Programmierung erfolgt in Bild 4a durch das Verbinden oder Nichtverbinden des Basisanschlusses mit der jeweiligen Wortleitung. Nach Bild 4a würde das unter der Adresse 0 gespeicherte Wort in der Gesamtschaltung von Bild 3 mit der Bitfolge 1011 beginnen. Die Programmierung erfolgt durch eine spezielle Metallisierungsmaske, welche die kritischen Verbindungen jeweils enthält oder nicht enthält. Bei MOS-

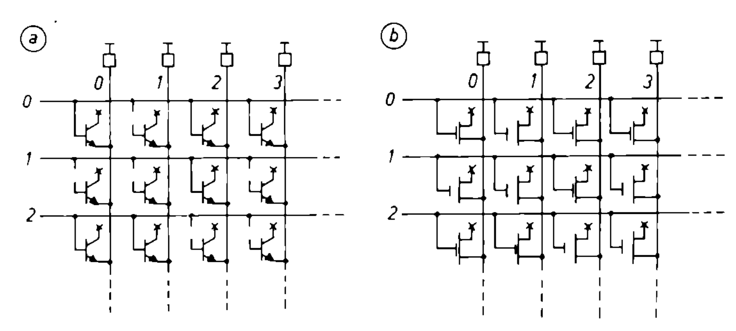


Bild 4 Festwertspeicherebene mit a) Bipolartransistoren und b) MOS-Feldeffekttransistoren als Speicherzellen
(× ^ Anschlüsse an die «Plusspannung»)

ROMs bestehen die Speicherzellen ausschliesslich aus MOS-Transistoren. Bild 4b zeigt eine entsprechende Schaltung. Die Programmierung wird hier im allgemeinen nicht durch den Anschluss oder Nichtanschluss der Gate-Elektrode vorgenommen, sondern durch die Dicke der Gate-Oxidschicht. Ist diese dünn, so beträgt die Schwellenspannung des Transistors nur einige Volt. Sie wird im aktiven Zustand der Wortleitung sicher überschritten, so dass der Transistor leitet und die entsprechende Bitleitung mitzieht. Bei einer dicken Oxidschicht liegt die Schwellenspannung dagegen weit über der Speisespannung des Speichers, so dass sie überhaupt nicht erreicht werden kann, der Transistor somit immer gesperrt bleibt.

Zu den gezeigten Speicherzellenanordnungen bestehen noch verschiedene Variationsmöglichkeiten, einige findet man in Bild 5, [4], dargestellt.

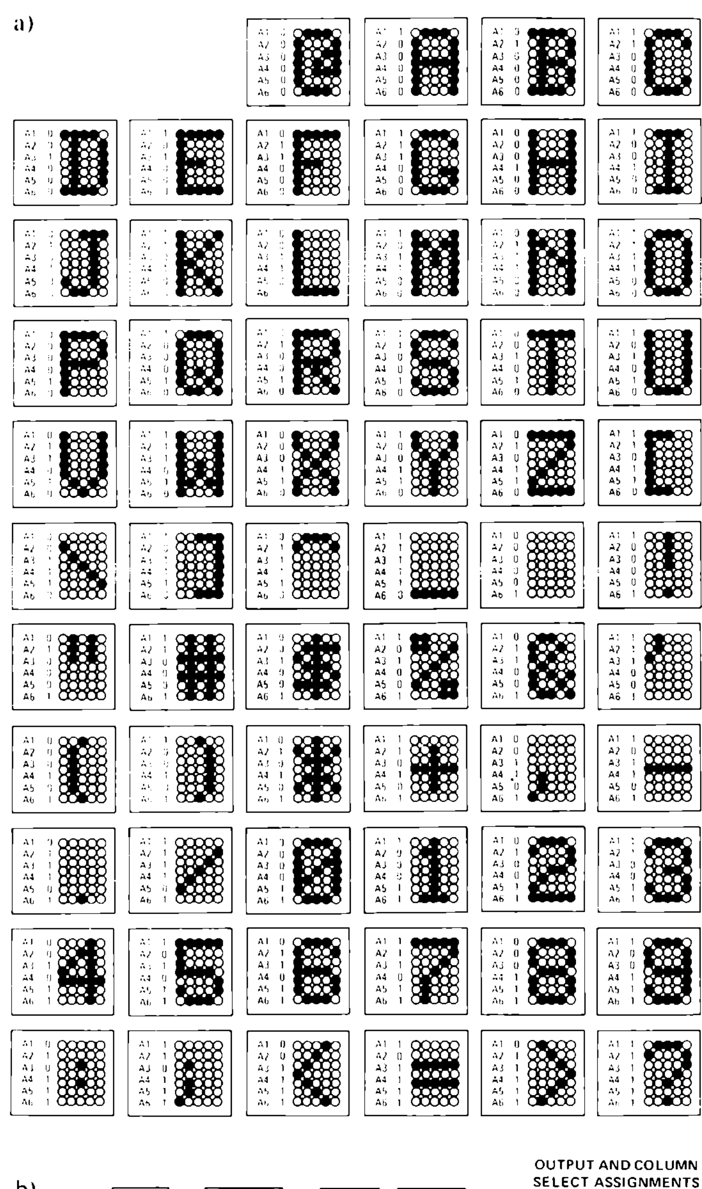
Bei dem nach der Herstellung programmierbaren Festwertspeicher (field programmable ROM) werden in Bipolartechnik die gleichen Dioden- oder Transistorspeicherzellen verwendet wie beim gewöhnlichen ROM, mit dem Unterschied, dass alle Verbindungen zu den Wort- und Bitleitungen hergestellt werden. Jeweils ein Anschluss pro Speicherzelle (bei Transistoren normalerweise der Emitteranschluss) wird jedoch mit sehr kleinem Leiterquerschnitt gebaut. Damit sind im Neuzustand alle Speicherzellen je nach dem Aufbau des Speichers entweder alle auf 0 oder auf 1 (wie zum Beispiel bei der Anordnung nach Bild 4) eingestellt. Wird nun vom Anwender in einer Zelle die inverse Information gewünscht, so wird zur Programmierung ein starker Stromimpuls durch die entsprechende Speicherzelle geschickt, wodurch die Verbindung mit dem kleinen Querschnitt örtlich thermisch überlastet wird und wie eine Schmelzsicherung verdampft und damit aufgetrennt wird. Anstelle des Aufbrennens von Verbindungsleitungen wird von einigen Herstellern ein Verfahren angewendet, bei welchem ein normalerweise gesperrter PN-Übergang durch Überspannungseinwirkung zum irreversiblen Durchbruch gebracht wird.

Die Speicherzellen eines nur in MOS-Technik herstellbaren lösch- und damit wieder programmierbaren Festwertspeichers (RePROM) bestehen aus Feldeffekttransistoren, deren Schwellenspannung wie beim gewöhnlichen MOS-ROM die Zelleninformation bestimmt. Die Einstellung der Schwellenspannung erfolgt jedoch nur elektrisch und reversibel durch das Einbringen einer Ladung auf eine vollständig isolierte Si-Gate-Elektrode. Dies geschieht durch kurzzeitiges Anlegen einer hohen Spannung an die Drain-Elektrode. Gelöscht werden diese Speicher durch Bestrahlung mit ultraviolettem Licht (Quarzlampe) durch ein im IC-Gehäuse eingebautes Quarzglasfenster.

6.3 Anwendungen

Das grösste Anwendungsgebiet der Festwertspeicher liegt zweifellos im Bereich der Computertechnik zur Speicherung fester Programme. Dies können zum Beispiel die zum Laden des Speichers eines Rechners mit dem zur Ausführung bestimmten Programm notwendigen Maschinenbefehle sein oder kleine Programmteile, sogenannte Mikroprogramme, welche den Programmierer davon befreien, ständig vorkommende Operationen immer wieder neu zu programmieren, zum Beispiel die vier mathematischen Grundoperationen ([5], Abschnitt 5.4). In Mikroprozessoren (Mikrocomputern) schliesslich, welche fest verdrahtet die Steuerung automatischer Anlagen übernehmen, wie etwa die Verkehrsregelung auf ganzen Strassenzügen oder Plätzen, sind meistens die gesamten zur Erfüllung der Aufgabe notwendigen Programme in Festwertspeichern untergebracht. Bei allen diesen Anwendungen ist es kaum möglich, ein Programm zu entwickeln und nachher ohne Änderungen oder Korrekturen im Endprodukt zu verwenden. Auch in der ersten Betriebszeit werden meistens noch Änderungen oder Korrekturen nötig sein. Aus diesen Gegebenheiten hat sich

a)



b)

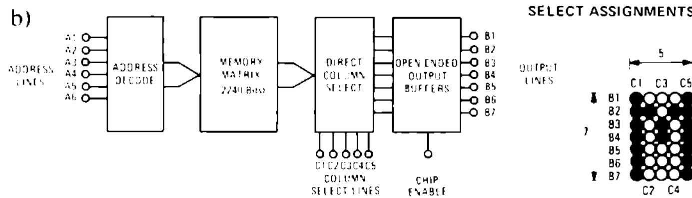


Bild 5 Als Zeichengenerator programmierter Festwertspeicher MCM 1131 von Motorola
a) Die damit darstellbaren Zeichen, Buchstaben und Zahlen des ASCII-Codes
b) Blockschaltung des Speichers und Bezeichnung der Anzeigematrix

die heute allgemein verbreitete Strategie des stufenweisen Übergangs zwischen den verschiedenen Festwertspeichertypen entwickelt: Im Entwicklungsstadium werden zur Speicherung RePROMs eingesetzt, in welchen die fehlerhaften Programme einfach gelöscht und korrigiert wieder eingeschrieben werden können. Bei der praktischen Erprobung in einer Kleinserie werden dann die billigeren PROM-Typen eingesetzt, welche immer noch ohne allzu grosse Verluste durch anders programmierte Exemplare ersetzt werden können. Zu den erst bei grossen Stückzahlen wirtschaftlichen herstellerprogrammierten ROMs wird erst übergegangen, wenn mit Sicherheit keine weiteren Korrekturen zu erwarten sind und die Stückzahlen entsprechend hoch sind.

Ein weiteres grosses Anwendungsgebiet ist die Code-Umsetzung und Zeichenerzeugung für elektronische Anzeigen. Da sich dabei immer wieder die gleichen Probleme stellen, werden verschiedene entsprechend programmierte Speicher als Katalogtypen bei vielen Herstellern geführt. Mit dem Motorola-ROM MCM 6561 mit einer Kapazität von 8192 bit können zum Beispiel die vier alphanumerischen Zeichencodes ASCII, Hollerith, Selectric und EBCDIC ineinander umgewandelt werden. Zur Anzeige alphanumerischer Zeichen werden neben Kathodenstrahlröhren häufig Leuchtdiodenmatrizen eingesetzt. Eine Anzahl Leuchtdioden, z.B. 35, bilden, eingeteilt in sieben Zeilen und fünf Kolonnen, eine Fläche, auf welcher durch eine entsprechende Ansteuerung der Dioden Buchstaben, Zahlen und Zeichen dargestellt werden können. Bild 5a zeigt die auf diese Weise erzeugten Zeichen des ASCII-Codes. Bei jedem Zeichen, Buchstaben oder jeder Zahl ist jeweils das entsprechende Codewort mit den Bits A_1 bis A_6 angegeben. Die Informationen, welche der 35 Dioden zur Darstellung der einzelnen Zeichen aktiviert werden müssen, können sehr bequem in einem ROM gespeichert werden. Viele Hersteller, unter anderem Motorola und Texas Instruments, führen entsprechend programmierte Typen. (Neue Typen arbeiten auch mit 7×9- oder 7×12-Dioden-Anordnungen.) Bild 5b zeigt den Aufbau des bereits klassischen Motorola-Zeichengenerator-ROM MCM 1132. Mit den sechs Adressenbits A_1 bis A_6 wird das gewünschte der 2^6 = 64 Zeichen gewählt. Am Ausgang der Speicherebene mit 2240 Speicherzellen erscheint die Zeicheninformation als 35-bit-Wort (2240:64). Jedes Bit enthält für eine der 35 Dioden die Information hell oder dunkel in Form einer logischen 1 oder 0. Jeweils 7 bit dieses Wortes bilden die Information für eine der 5 Kolonnen C_1 bis C_5 der Anzeige, welche einzeln durch die Ansteuerung des entsprechenden Kolonnenwahleinganges (Column select line C_1 bis C_5) auf die Ausgänge B_1 bis B_7 des Speichers geschaltet werden können. Die einzelnen Zeichen werden auf diese Weise zeitlich zerlegt in Kolonnen dargestellt. Durch diesen Zeitmultiplexbetrieb kommt man mit einem Minimum an Verbindungsleitungen zwischen Speicher und Anzeigefeld aus. Als Kuriosum soll noch erwähnt werden, dass Motorola auch mit griechischen und japanischen Buchstaben programmierte ROMs zur Zeichenerzeugung katalogmässig liefert!

Anhand des einfachen Beispiels eines Decoders zur Ansteuerung von Siebensegmentanzeigen soll nun noch gezeigt werden, wie die ROM-Technik auch beim Arbeiten mit gewöhnlichen logischen Schaltungen vorteilhaft sein kann. Bild 6 zeigt die zehn Dezimalziffern in der Siebensegmentdarstellung und die Bezeichnung der Segmente. Der Decoder hat die Aufgabe, für jede im BCD-Code eingegebene Dezimalziffer die Steuersignale für die sieben Segmente der Anzeige zu bilden. Die Verknüpfungsregeln sind durch die mit Tabelle 2 gegebene Wahrheitstabelle (abgeleitet aus Bild 6)

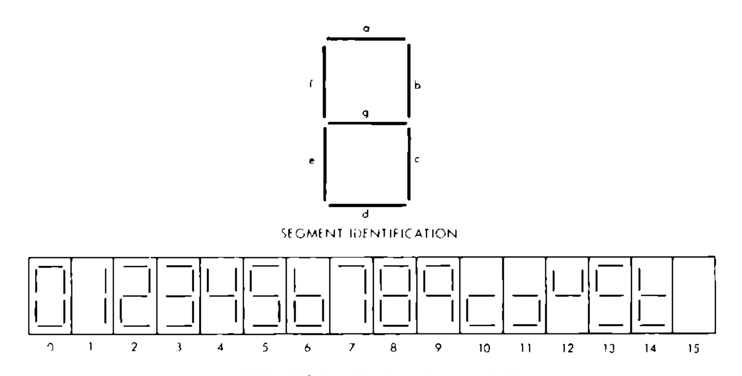


Bild 6 Siebensegmentdarstellung der Dezimalziffern 0 bis 9 und Bezeichnung der Segmente

Tabelle 2
Wahrheitstabelle eines BCD-Siebensegmentdecoders

Dezimal-Ziffer	*D*	*C*	*B*	*A*	*a*	*b*	*c*	*d*	*e*	*f*	*g*
	BCD-Code				7-Segment-Decodierung						
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	0	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	0	0	1	1

definiert. In der klassischen Realisierungsweise würde nun für jedes der sieben Ausgangssignale *a* bis *g* die logische Verknüpfungsfunktion mit den vier Eingangssignalen *A* bis *D* anhand der Wahrheitstabelle aufgestellt, diese mit einem der bekannten Verfahren minimisiert und durch Torschaltungen realisiert. Eine entsprechende Schaltung, wie sie im integrierten Baustein SN 7449 von TI verwendet wird, zeigt Bild 7. Der Schaltungsaufwand beträgt ohne die «blanking»-Schaltung 24 Tore. Bei der Realisierung als ROM werden die Eingangssignale *A* bis *D* dem Adressendecoder zugeführt, an dessen Ausgängen 0 bis 9 die 10 Wortleitungen der Speicherebene angeschlossen sind. An den 70 Kreuzungspunkten mit den 7 (*a* bis *g*) Bitleitungen werden die Speicherzellen entsprechend dem rechten Teil der Wahrheitstabelle programmiert. Eine mögliche Schaltung zeigt Bild 8. Entgegen der Darstellung in Bild 3, wird hier eine invertierte Decoderfunktion vorausgesetzt: Der aktivierte Ausgang soll auf Massepotential liegen, alle andern Ausgänge liegen hoch. Dadurch wird auch die Funktion der Speicherzellen umgekehrt, eine vorhandene (oder beidseitig angeschlossene) Diode entspricht jetzt einer logischen Null. Diese Anordnung wurde gewählt, weil die Wahrheitstabelle bedeutend weniger Nullen als Einsen enthält und damit die Zahl der Dioden kleiner wird. Diesen Vorteil kann man in einem diskreten Aufbau ausnützen. Betrachtet man den Speicher von der Ausgangsseite her, so ergibt sich zum Beispiel für den Ausgang *a* = 1 die Bedingung, dass die Wortleitungen 1 und 4 und 6 hoch

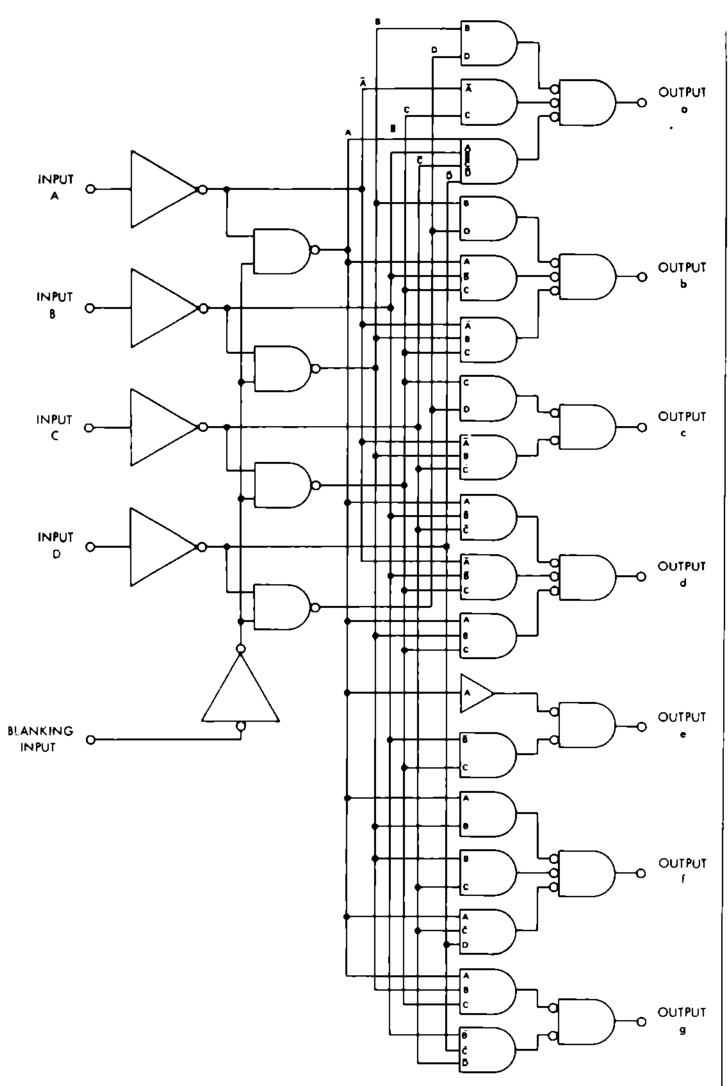


Bild 7 Schaltung des BCD-Siebensegmentdecoders SN 7449 von Texas Instruments

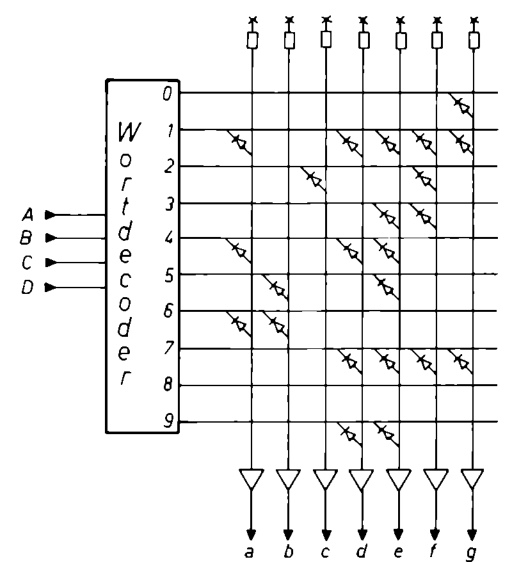


Bild 8 Als BCD-Siebensegmentdecoder programmierter Festwertspeicher (Wortdecoder mit aktivem Ausgang tief)

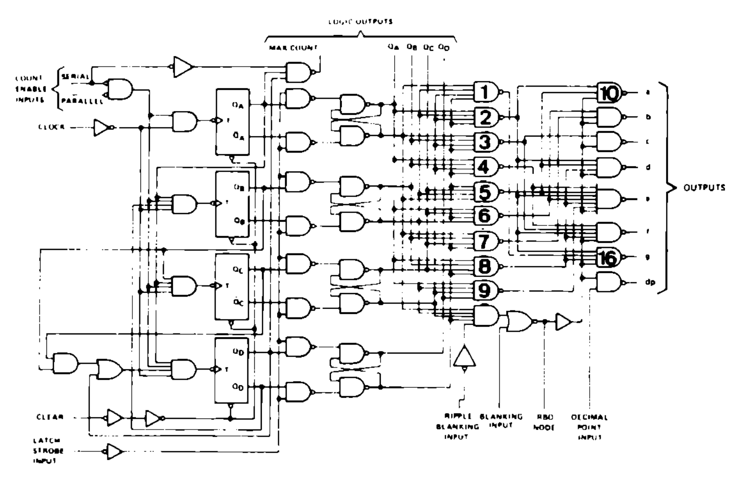


Bild 9 Schaltung des MSI-Bausteins SN 74143 mit in ROM-Technik aufgebautem BCD-Siebensegmentdecoderteil (NAND-Tore 1 bis 16)

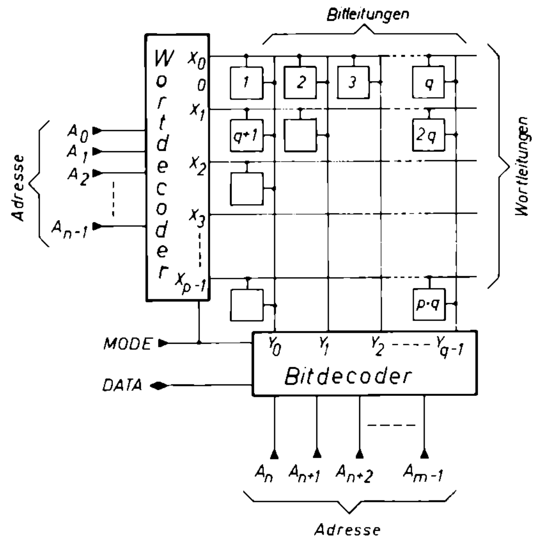


Bild 10 Aufbau eines RAM mit wortorganisierter Speicherebene

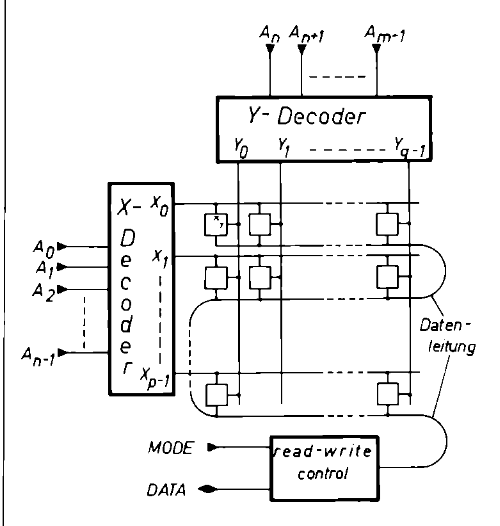


Bild 11 Aufbau eines RAM mit bitorganisierter Speicherebene

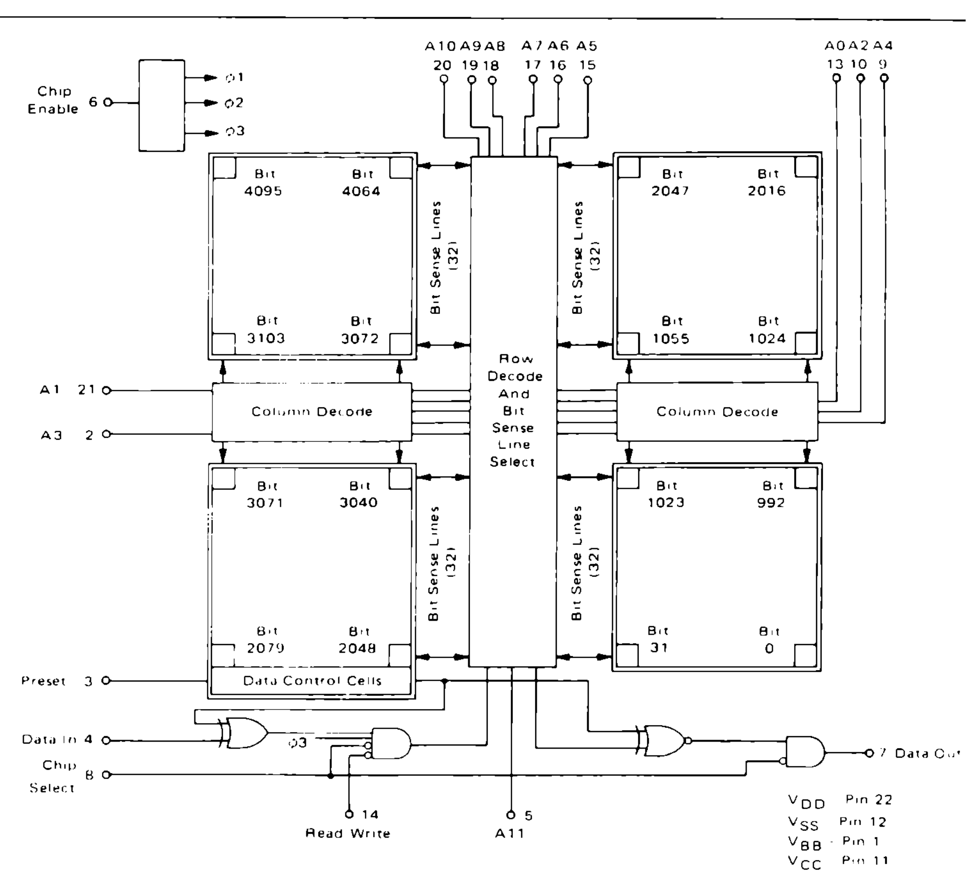


Bild 12 Aufbau des dynamischen RAM MCM 6605 von Motorola mit einer Kapazität von 4096 bit, aufgeteilt in vier wortorganisierte Speicherebenen zu je 1024 bit

liegen müssen. Diese Verknüpfung kann nun jedoch auch ganz konventionell mit einem UND-Tor hergestellt werden, dessen drei Eingänge an den Wortleitungen 1, 4 und 6 angeschlossen werden. Um alle sieben Ausgangsfunktionen auf diese Weise herzustellen, werden 7 UND-Tore mit 1 bis 6 Eingängen ($a \to 3$, $b \to 2$, $c \to 1$, $d \to 4$, $e \to 6$, $f \to 4$, $g \to 3$) benötigt. Verwendet man die in der TTL-Technik üblicheren NAND-Tore, so erhält man mit einem Aufwand von 3 IC-Gehäusen die invertierten Ausgangsfunktionen. Da der Wortleitungsdecoder als vollständig integrierte Schaltung eingesetzt werden kann (z. B. SN 74151), ist der ganze BCD-7-Segment-Decoder in ROM-Struktur mit nur vier IC-Gehäusen realisiert. Es bleibt dem Leser überlassen, aus dem Bild 7 die zur Realisierung nach der klassischen Methode notwendigen Bausteine herauszuzählen, sicher sind es bedeutend mehr. Da mit diesem Verfahren auch die ganze Arbeit der Minimisierung und Adaption an die erhältlichen Logikbausteine wegfällt, ist es vor allem zur Realisierung kombinatorischer Schaltungen im Labor geeignet. Dass jedoch auch in der Grosserienanfertigung integrierter Schaltungen davon Gebrauch gemacht wird, zeigt Bild 9. In der LSI-Schaltung SN 74143/144 (4-bit-Zähler + Speicher + 7-Segment-Decoder) ist der Sieben-Segment-Decoderteil in ROM-Technik aufgebaut. Die NAND-Tore 1 bis 9 bilden den Adressendecoder (die Zahl 8 wird nicht decodiert, da das zugehörige Ausgangssignal aus lauter Einern besteht), die Ausgangs-NAND-Tore sind mit 10 bis 15 bezeichnet (die Ausgangssignale sind gegenüber der Wahrheitstabelle von Tabelle 2 invertiert, und bei den Zahlen 6 bzw. 9 werden zusätzlich die Segmente a bzw. d zur besseren Lesbarkeit aktiviert). Gegenüber der konventionellen und älteren Decoderschaltung des IC SN7449 von Bild 7 werden damit 9 Tore eingespart!

7. RAM

7.1 Aufbau

Der Aufbau eines RAM unterscheidet sich vom ROM neben einer vollständig anderen Struktur der Speicherzellen durch die Existenz eines zweiten Decoders, des Kolonnen- oder Bitdecoders. Zwei verschiedene Formen des Aufbaus sind gebräuchlich. Die erste Bauform mit einer wortorganisierten Speicherebene geht direkt aus der ROM-Struktur von Bild 3 oder 8 durch die Zuschaltung des Bitdecoders an die Bitleitungen hervor und ist in Bild 10 dargestellt. Durch einen ersten Teil der Adresse (A_0 bis A_{m-1}) wird über den Wortdecoder eine der $p = 2^n$ Wortleitungen (X_0 bis X_{p-1}) aktiviert (zum Beispiel auf Logisch-1-Potential gebracht, während alle andern auf 0 bleiben). Dadurch werden die daran angeschlossenen Speicherzellen mit den $q = 2^{m-n}$ Bitleitungen verbunden. Damit erscheint beim Lesen an den Anschlüssen Y des Bitdecoders der Inhalt der durch den Wortdecoder ausgewählten Speicherzellenzeile der Speicherebene. Durch den zweiten Teil der Adresse (A_n bis A_{m-1}) wird nun daraus über den Bitdecoder das gewünschte Bit auf den Datenanschluss DATA des Speichers geschaltet. Für die Umschaltung des Speichers von «Lesen» (read) auf «Schreiben» (write) ist der mit MODE bezeichnete Anschluss vorhanden. Beim Schreiben wird die am DATA-Anschluss anliegende Information auf eine über den Bitdecoder ausgewählte Bitleitung geschaltet und in die daran angeschlossene Speicherzelle der über den Wortdecoder ausgewählten Zeile der Speicherebene eingeschrieben. Meistens sind die Wort- und Bitleitungen zur Umschaltung der Speicherzellen auf «Schreiben» oder «Lesen» oder zur Übertragung der Daten und ihrer Komplemente zweidrähtig ausgeführt. Die Funktionsweise bleibt jedoch immer die gleiche: Durch die zweifache Decodierung der m-Bit-Adresse mit den Wort- und Bitdecodern ist es möglich, den Inhalt jeder der $p \cdot q = 2^m$ Speicherzellen auf den Datenanschluss zu schalten oder die dort angelegte Information darin einzuschreiben. Häufig werden anstelle der Ausdrücke Wort- und Bitdecoder und Wort- und Bitleitungen die Bezeichnungen X- und Y-Decoder und X- und Y-Leitungen verwendet, wie bei der anschliessenden Vorstellung der zweiten RAM-Bauform.

Die zweite Form des Aufbaus mit einer bitorganisierten Speicherebene ist in Bild 11 wiedergegeben. Im Gegensatz zu Bild 10 wird hier eine spezielle (meistens zweidrähtige) Datenleitung verwendet, welche alle Speicherzellen mit der Schreib/Lese-Umschaltlogik (read-write control) verbindet. Die Speicherzellen sind nun so ausgelegt, dass sich nur diejenige effektiv an die Datenleitung ankoppelt, bei welcher die angeschlossene X- und Y-Leitung aktiviert ist. Durch den ersten Teil der Adresse (A_0 bis A_{n-1}) wird über den X-Decoder die Zeile, durch den zweiten Teil (A_n bis A_{m-1}) die Kolonne der Speicherebene, in der sich die gewünschte Speicherzelle befindet, ausgewählt. Die Schreib/Lese-Umschaltlogik be-

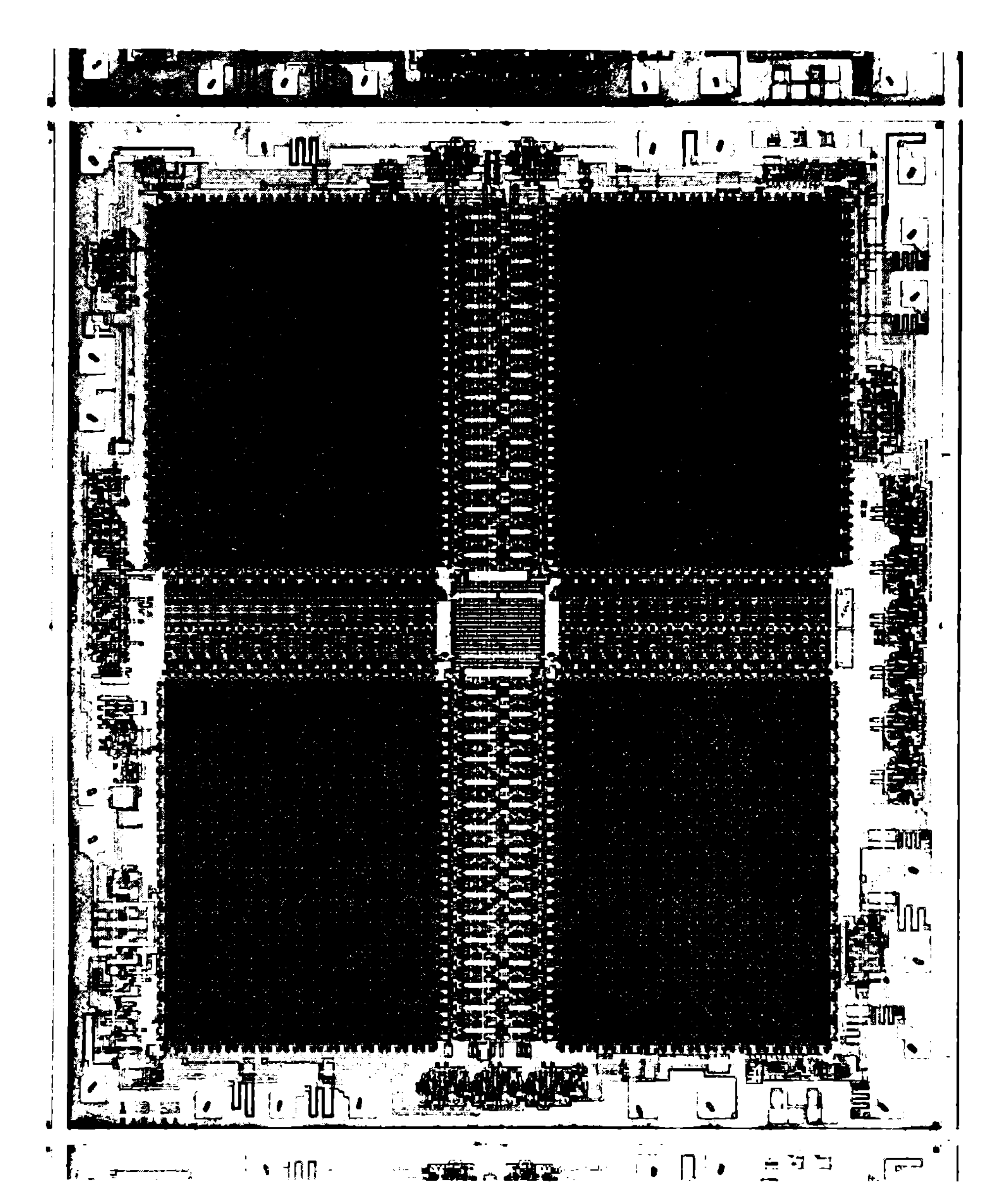


Bild 13 Makroaufnahme der Speicherschaltung MCM 6605 (Bild Motorola)

stimmt die Richtung des Informationsflusses vom Datenanschluss (DATA) über die Datenleitung in die gewählte Speicherzelle beim Schreiben oder umgekehrt beim Lesen.
Normalerweise wird noch zusätzlich wie beim ROM eine «Chip enable»-Funktion vorgesehen, um bei parallelgeschalteten Speichern jeden einzeln wählen zu können. Wie beim ROM werden bei RAMs grosser Speicherkapazität meistens auf diese Weise bereits auf einen Chip mehrere Speicherebenen angeordnet. Bild 12 zeigt dies am Beispiel des dynamischen Motorola-RAM MCM 6605 in N-MOS-Technik mit vier parallelgeschalteten Speicherebenen zu je 1024 bit. Bild 13 zeigt eine Grossaufnahme des Speicherchips. Deutlich zu sehen sind die vier Speicherebenen mit den Speicherzellen und kreuzförmig in der Mitte die Decoderschaltungen. Dieser Speicher entspricht dem letzten Stand der Entwicklung.

7.2 Speicherzellen

Die Speicherzellen eines RAM können auf sehr viele Arten realisiert werden, je nach der verwendeten Schaltungsart, Technologie und Betriebsart (statisch oder dynamisch) des Speichers. Es sollen hier nur einige wenige ausgewählte Beispiele besprochen werden, welche die grundsätzlichen Möglichkeiten zeigen.
Die statischen Speicherzellen, welche die Information unbegrenzt speichern können (vorausgesetzt, dass die Betriebsspannung nicht ausfällt), arbeiten immer mit einer bistabilen Flip-Flop-Schaltung. Bild 14a zeigt die einfachste Schaltungsmöglichkeit mit bipolaren Transistoren für den in Bild 10 gezeigten Speicheraufbau mit wortorganisierter Speicherebene. Die beiden übers Kreuz gekoppelten Transistoren T_1 und T_2 bilden mit den Kollektorwiderständen R_1 und R_2 ein Flip-Flop. Die Wortleitung W liegt im Ruhezustand auf Massepotential. Ist die Information 1 gespeichert, so ist T_1 leitend und T_2 gesperrt. Der Kollektorstrom von T_1 fliesst über die Wortleitung ab. Wird nun diese aktiviert (auf 1-Potential gehoben), so fliesst der Kollektorstrom von T_1 auf die Bitleitung B und zieht diese über den kleinen Lastwiderstand R_L ebenfalls etwas hoch. Dies wird im Bitdecoder als logische 1 interpretiert. Ist eine 0 gespeichert, so geht beim Lesen $\bar{B}$ hoch (was als 0 interpretiert wird) und B bleibt tief. Soll eine 0 in die Speicherzelle eingeschrieben werden, so wird bei aktivierter Wortleitung B auf Massepotential festgehalten und $\bar{B}$ hochgelegt. Dadurch wird der Zustand T_2 leitend T_1 gesperrt erzwungen, welcher auch nach dem Schreibvorgang erhalten bleibt. (Das Schreiben einer 1 wird mit $B = 0$ und $\bar{B} = 1$ durchgeführt.)
Für einen Speicher mit bitorganisierter Speicherebene (Bild 11) wird die Schaltung von Bild 14b verwendet. Sie unterscheidet sich von der ersten nur durch je einen zusätzlichen Emitter bei beiden Transistoren. Wenn mindestens eine der Adressierleitungen X und Y nicht aktiviert ist und damit auf Massepotential liegt, fliesst der Kollektorstrom des leitenden Transistors über diese ab. Nur wenn beide Leitungen X und Y hoch liegen, die Zelle sich also sowohl in der durch den X- als auch den Y-Decoder gewählten Zeile bzw. Kolonne befindet, findet eine Ankoppelung an die zweidrähtige Datenleitung D und d zum Lesen oder Schreiben statt.
Der Aufbau einer statischen MOS-Speicherzelle mit N-Kanal-

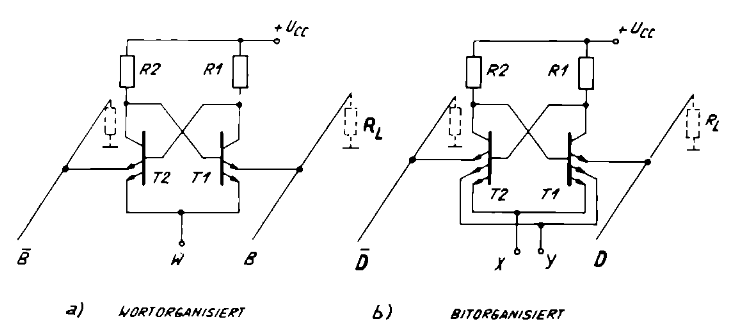


Bild 14 RAM-Speicherzellen mit Bipolartransistoren

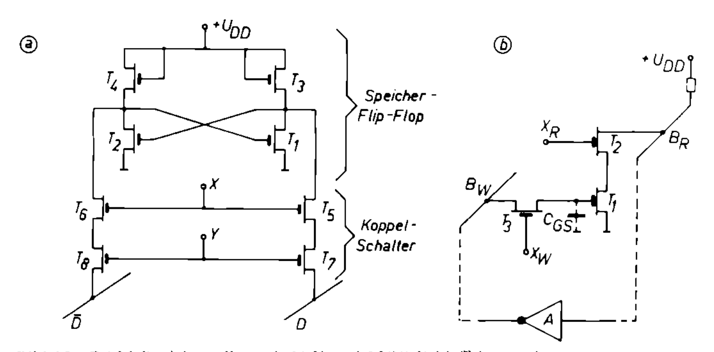


Bild 15 RAM-Speicherzellen mit N-Kanal-MOS-Feldeffekttransistoren
a) Statische 8-Transistor-Zelle für eine bitorganisierte Speicherebene
b) Dynamische 3-Transistor-Zelle für eine wortorganisierte Speicherebene mit Auffrischverstärker A

Transistoren vom Anreicherungstyp einer bitorganisierten Speicherebene ist in Bild 15a dargestellt. Die Transistoren T_1 und T_2 bilden wieder ein kreuzgekoppeltes Flip-Flop. Als Arbeits-Drain-Widerstände werden die Transistoren T_3 und T_4 eingesetzt (Widerstände wären in der MOS-Technik sehr schlecht herstellbar). Zur Ankoppelung der Zelle an die zweidrähtige Datenleitung (D und $\bar{D}$) werden die Transistoren T_5 bis T_8 als analoge Schalterpaare eingesetzt. Die Ankopplung an die Datenleitung ist nur dann möglich, wenn beide Koppelschalter-Transistorpaare leitend sind, also die X- und Y-Leitungen auf 1-Potential liegen.

Bei den dynamischen Speicherzellen, welche nur in MOS-Technik hergestellt werden, wird die Information als Ladung auf einem Kondensator gespeichert. Die Gate-Source-Kapazität eines FET liegt in der Grössenordnung 1 pF. Da unvermeidliche Leckströme (im allgemeinen Sperrströme parasitärer Dioden in der Grössenordnung von 100 pA) die Speicherkondensatoren dauernd entladen, muss die Information periodisch (z.B. jede ms) aufgefrischt werden. Dies geschieht je nach Zellentyp zeilenweise automatisch durch periodisches Durchlaufen aller Adressenkombinationen des Wortdecoders bei einem Speicheraufbau nach Bild 10 (z.B. MCM 6605) oder über einen speziellen Anschluss (Refresh) für den ganzen Speicher gleichzeitig (z.B. Intel 2105). Die Vorteile der dynamischen Zellen liegen in einem bedeutend geringeren Leistungsverbrauch und einem kleineren Flächenbedarf. Eine weitverbreitete dynamische 3-Transistor-Zelle für den Einsatz in einer wortorganisierten Speicherebene (Bild 10) zeigt Bild 15b. Die Information wird auf der Gate-Source-Kapazität C_{GS} des Transistors T_1 gespeichert. Zum Einschreiben wird die Wortleitung X_W auf 1-Potential gelegt, wodurch T_3 leitet und die Information von der Bitleitung B_W auf den Kondensator übertragen wird. Zum Lesen wird die zweite Wortleitung X_R hochgelegt, so dass T_2 leitet. Wurde in der Zelle eine 1 gespeichert, so ist C_{GS} positiv geladen, T_1 leitet ebenfalls, und die Bitleitung B_R wird auf Massepotential gezogen. Diese Spannungsänderung wird im Bitdecoder als 1 interpretiert. Beim Speicherinhalt 0 ist C_{GS} nicht geladen, T_1 gesperrt, und B_R liegt hoch. Zum Auffrischen der Information werden beide Bitleitungen aktiviert, wodurch die ausgelesene Information über den für jede Kolonne einmal im Bitdecoder vorhandenen Auffrischungsverstärker A wieder neu eingelesen wird.

Die Vorteile gegenüber der statischen Schaltung von Bild 15a sind offensichtlich: Anstatt acht Transistoren werden nur drei benötigt, und die Speicherzelle verbraucht im Ruhezustand keine Leistung. Dagegen muss natürlich durch zusätzliche Schaltungen für das periodische Auffrischen gesorgt werden. Zudem kann während der allerdings kurzen Auffrischzeiten (Grössenordnung 100 ns pro Zeile) der Speicher nicht benützt werden.

8. Schieberegister

8.1 Aufbau

Im Gegensatz zu den vorhergehenden beiden Speicherarten sind beim Schieberegister die Speicherzellen nur eindimensional in einer Linie angeordnet. Dadurch wird das Verbindungsproblem drastisch vereinfacht, indem jede Speicherzelle ihre Information nur an die folgende abgeben oder von der vorhergehenden aufnehmen können muss. Das ganze System von mehrdrähtigen Wort-, Bit- und Datenleitungen wird ersetzt durch zwei (teilweise 3) Taktleitungen, welche die Weitergabe der Informationen in der ganzen Speicherzellenkette steuern. Den grundsätzlichen Aufbau eines SR zeigt Bild 16. Jede der n Speicherzellen (n = Registerlänge) besteht aus zwei Stufen, einem Master-Speicher M und einem Slave-Speicher S (gleiche Anordnung wie bei einem Master-Slave-Flip-Flop). Alle Stufen sind miteinander durch Koppelschalter verbunden. Diejenigen, welche die beiden Stufen einer Zelle miteinander verbinden, werden durch den Takt Φ_2 gesteuert, die übrigen (zwischen den Stufen verschiedener Zellen) durch Φ_1. Die Informationsverschiebung geht nun so vor sich, dass in einer ersten Phase die durch Φ_1 gesteuerten Schalter geschlossen werden, wodurch die Master-Stufe jeder Zelle die Information der Slave-Stufe der vorhergehenden übernimmt. In der zweiten Phase werden die durch Φ_2 gesteuerten Schalter geschlossen, und die Information wird innerhalb jeder Zelle von der Master- auf die Slave-Stufe übertragen. Die Notwendigkeit dieser zweistufigen Anordnung kann man sich sehr schön am Beispiel der Gepäcktransportkette (Abschnitt 2.3) überlegen: ein Passagier (einer Speicherstufe entsprechend) kann nicht gleichzeitig ein Gepäckstück (die Information) an den nächsten weitergeben und ein neues empfangen. Als Ergänzung, welche in den meisten Schieberegistern fest eingebaut ist, befindet sich vor der ersten Speicherzelle die Recirculate-Logik. Durch den Steuereingang REC (recirculate control oder stream select) kann gewählt werden, ob neue, am Dateneingang DATA IN anliegende Informationen eingelesen werden sollen oder ob die alten, am Datenausgang DATA OUT angekommenen wieder eingelesen werden sollen. Neuere Schieberegister besitzen zudem fast durchwegs einen eingebauten Generator zur Erzeugung der beiden Taktsignale Φ_1 und Φ_2 aus einem einzigen von aussen zugeführten Taktsignal.

Längere Schieberegister (mit Kapazitäten von über 16 bit) werden ausschliesslich in MOS- (oder auch CMOS-)Technik hergestellt.

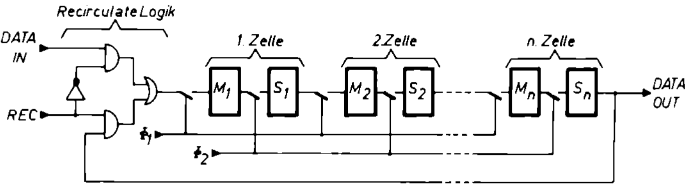


Bild 16 Aufbau eines Schieberegisters mit Zweiphasen-Taktsteuerung

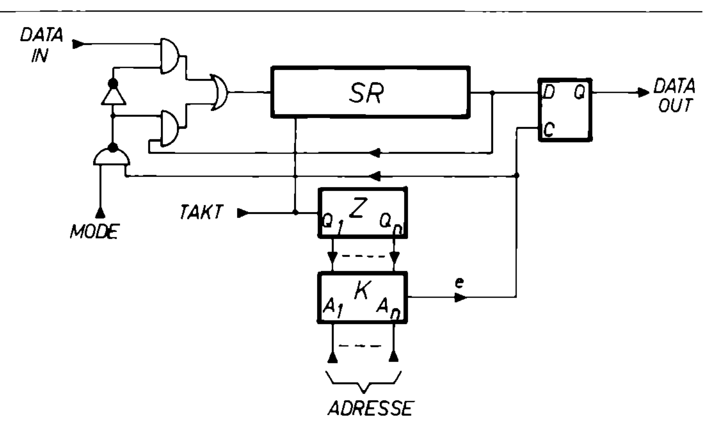


Bild 18 Schaltung zum praktischen Betrieb eines dynamischen Schieberegisters

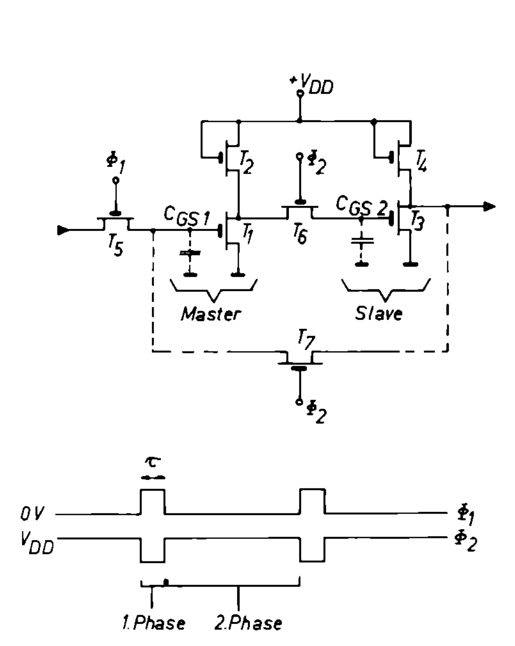


Bild 17 Dynamische bzw. halbstatische Schieberegister-Speicherzelle mit Zweiphasen-Taktsteuerung mit N-Kanaltransistoren

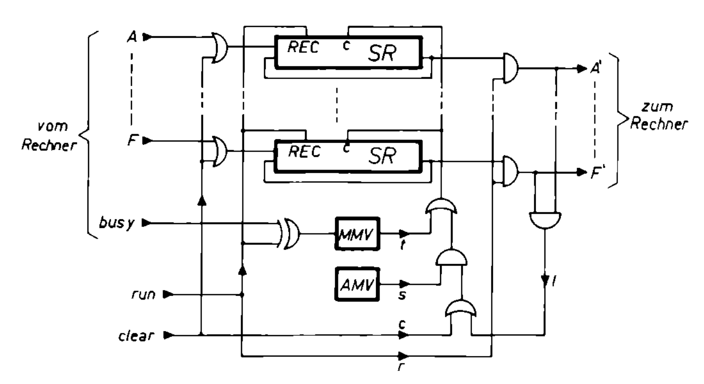


Bild 19 Ein mit Schieberegister aufgebauter Programmspeicher zu einem elektronischen Tischrechner

Wie beim RAM, können die Speicherzellen statisch oder dynamisch aufgebaut werden. Zusätzlich besteht noch die häufig ausgenützte Kombination der beiden Verfahren, indem die Speicherung nur in einer der beiden Phasen statisch erfolgt, in der anderen dagegen dynamisch. Damit lässt sich der Schaltungsaufwand reduzieren. Bei diesen ebenfalls als statisch bezeichneten Registern müssen lediglich gewisse Bedingungen an die Taktimpulsform eingehalten werden, damit die Information auch beim Stillstand nicht verlorengeht.

8.2 Speicherzellen

Eine dynamische Speicherzelle vom Ratio- oder Verhältnistyp (bestimmte Verhältnisse zwischen den Transistorgeometrien müssen eingehalten werden) zeigt Bild 17. (Die gestrichelte Verbindung über T_7 ist vorerst wegzudenken.) Die Transistoren T_1 und T_2 bilden mit der Gate-Source-Kapazität C_{GS1} die Master-Stufe, T_3 und T_4 mit C_{GS2} die Slave-Stufe. In der ersten Taktphase (Φ_1 hoch) wird die Information der vorausgehenden Zelle durch den mit T_5 realisierten Koppelschalter auf die Speicherkapazität der Master-Stufe übertragen. Am Source-Anschluss von T_1 erscheint das invertierte Signal (T_1 bildet mit T_2 als Arbeitswiderstand einen Inverter). Dieses wird in der zweiten Phase (Φ_2 hoch) über den mit T_6 realisierten Koppelschalter auf die Speicherkapazität der Slave-Stufe kopiert und erscheint am Source-Anschluss von T_3 nochmals invertiert (also wieder richtig) als neues Ausgangssignal der Speicherzelle. Durch die Einführung eines einzigen zusätzlichen Koppelschalters mit dem Transistor T_7, welcher den Eingang in der zweiten Phase mit dem Ausgang verbindet, wird in dieser ein statischer Speicherzustand geschaffen. Die Transistorpaare T_1, T_2 und T_3, T_4 bilden dann über T_6 und T_7 kreuzgekoppelt ein konventionelles Flip-Flop, welches die Zelleninformation beliebig lange halten kann. Damit in der dynamischen Phase 1 die Information nicht durch Entladung der Speicherkondensatoren verlorengeht, muss bei langsamem Betrieb lediglich darauf geachtet werden, dass die Taktimpulsbreite τ einen vom Hersteller vorgeschriebenen Maximalwert (in der Grössenordnung 10 μs) nicht überschreitet.

8.3 Anwendungen

In dynamischen Schieberegistern muss die Information ständig mit einer minimalen Taktfrequenz (Grössenordnung 10 kHz) verschoben werden. Im allgemeinen wird dabei die Ausgangsinformation über die Recirculate-Logik wieder auf den Eingang zurückgeführt, so dass die Information ständig in einem geschlossenen Kreis rundumläuft. Um dabei die Übersicht nicht zu verlieren, muss ein dynamischer Schieberegisterspeicher mit einigen Zusatzschaltungen ergänzt werden. Bild 18 zeigt den grundsätzlichen Aufbau. Die dem SR der Länge 2^n aus einem Taktgenerator ständig zugeführten Taktimpulse werden in einem n-bit-Zähler Z gezählt. Dieser läuft synchron mit dem SR, d.h. bei einem bestimmten Zählerstand befindet sich immer die gleiche Information am Ausgang des Registers. Der Zählerstand wird laufend in einem n-bit-Komparator K mit der eingestellten Adresse verglichen. Wenn beide übereinstimmen, erzeugt der Komparator ein Ausgangssignal e, wodurch die gewünschte, sich in diesem Moment gerade am SR-Ausgang befindende Information in ein D-Flip-Flop eingelesen wird, an dessen Ausgang DATA OUT sie nun unabhängig vom Registertakt weiter verwendet werden kann. Zum Einlesen neuer Informationen wird der Schreib/Lese-Steueranschluss MODE auf 1-Potential gelegt, wodurch der Registereingang kurzzeitig über die Recirculate-Logik mit dem Dateneingang DATA IN verbunden wird, sobald Zählerstand und Adresse übereinstimmen. In dieser Anordnung kann das SR wie ein RAM betrieben werden, die Zugriffszeit wird jedoch bei langen Registern sehr gross, im Mittel $(n/2) \cdot T$, wenn T die Taktperiode des SR ist. Bis vor wenigen Jahren wurde dieses Verfahren sehr häufig benützt, als mit RAMs noch nicht die heute üblichen grossen Speicherkapazitäten erreicht werden konnten, wohl aber mit Schieberegistern.

Das sinnvollste Anwendungsgebiet der Schieberegister liegt zweifellos dort, wo die Informationen immer in der gleichen Reihenfolge benötigt werden. Dies ist zum Beispiel in einfachen Rechenprogrammen der Fall. Für diese Anwendung wurde vom Verfasser ein Programmspeicher zu einem elektronischen Tischrechner Sharp PC 1001 entwickelt, welcher die Speicherung von bis zu 128

Programmschritten in sechs statischen Schieberegistern erlaubt. Damit können bereits recht umfangreiche Formeln, beispielsweise zur Berechnung von Tabellen, Kennlinien und Übertragungsfunktionen, programmiert und anschliessend beliebig oft automatisch durchgerechnet werden. Da dieser Rechner bereits intern programmierbar ist, allerdings mit einer im praktischen Gebrauch unzureichenden Programmkapazität, konnten verschiedene interne Funktionen für den Zusatzspeicher einfach übernommen werden. So etwa die Halt-Funktion zur Unterbrechung des Programmes oder das «busy»-Signal, welches die Ausführung eines Befehles anzeigt. Die vereinfachte Schaltung des Speichers zeigt Bild 19. Jedes Drücken einer der 30 Eingabetasten wird im Rechner in ein 6-bit-Wort codiert (30 der total möglichen 64 Bitkombinationen stellen damit je eine Tastenfunktion dar). Diese 6-bit-Worte werden beim Programmieren («run»-Leitung tief) über die Leitungen A bis F den Eingängen der sechs Schieberegister zugeführt. Bei jedem Tastendruck geht die «busy»-Leitung kurzzeitig hoch (der Rechner verarbeitet den eingegebenen Befehl) und löst mit der positiven Flanke über einen monostabilen Multivibrator MMV einen SR-Taktimpuls aus. Dadurch wird der Befehl in den Programmspeicher eingelesen. Zum Löschen wird bei hoch liegender «clear»-Leitung ein schnelles im astabilen Multivibrator AMV erzeugtes Taktsignal (100 kHz) auf die SR gekoppelt, und über die Eingangs-OR-Tore werden lauter Einer eingelesen (das Befehlswort 111111 wird vom Rechner ignoriert und dient damit als Leerzeichen). Zum Laufenlassen des gespeicherten Programms werden die SR über die nun hoch liegende «run»-Leitung auf Recirculate-Betrieb umgestellt und die Ausgänge über die Ausgangs-AND-Tore auf die zum Rechner gehenden Leitungen A′ bis F′ geschaltet. Jede fallende Flanke des «busy»-Signals erzeugt nun (invertiert über das EX-OR-Tor), sobald der vorherige Befehl vom Rechner ausgeführt wurde, über den MMV einen SR-Taktimpuls, wodurch der nächste Befehl am Ausgang erscheint und auf den Rechner gegeben wird. Die normalerweise am Schluss des Programms noch vorhandenen Leerzeichen werden durch ein AND-Tor an den Ausgängen A′ bis F′ detektiert, der schnelle Taktgenerator wird an die SR angekoppelt und der leere Teil damit rasch durchgeschoben, bis wieder der Programmanfang am Ausgang erscheint.

9. Produkte

Digitale Halbleiterspeicher werden heute von fast allen Herstellern integrierter Schaltungen in einer Vielzahl von Typen und Ausführungen angeboten. Dabei sind viele gegeneinander austauschbar, obwohl noch keine etwa mit der TTL-Logikfamilie 74-- vergleichbare Typennormung besteht. Eine vollständige Präsentation der erhältlichen Schaltungen müsste daher die einzelnen Typen aller Hersteller umfassen und würde den Rahmen dieser Einführung in die digitale Halbleitertechnik bei weitem sprengen. Um dem Leser trotzdem einen Eindruck des Angebotes zu vermitteln, sind in der Tabelle 3 jeweils einige willkürlich ausgewählte Typen der verschiedenen Speicherarten zusammengestellt.

Literaturverzeichnis

[4] J. H. Kirchner, «Halbleiterspeicher», «Der Elektroniker», Nr. 4 1972
[5] K. Wüthrich, «Mikrocomputer», «Der Elektroniker», Nr. 4 1974

Tabelle 3 Einige Beispiele digitaler Halbleiterspeicher

Speicherart	Bezeichnung/Hersteller	Organisation, Bitzahl	Schaltungsart	Zugriffszeit bzw. Schiebefrequenz	Technologie
ROM	Intel 3301	256 × 4 = 1024	TTL	25 ns	Schottky
ROM	NS MM 4233	512 × 8 = 4096	P-MOS	1 μs	Silicon Gate
ROM	Motorola 6560	1024 × 8 = 8192	N-MOS	225 ns	Metal Gate
PROM	Intersil 5600	32 × 8 = 256	TTL	40 ns	bipolar
PROM	Intel 3601	256 × 4 = 1024	TTL	60 ns	Schottky
PROM	Harris HPROM 2048	512 × 4 = 2048	TTL	50 ns	bipolar
Re PROM	NS MM 4203	256 × 8 = 2048	P-MOS	1 μs	Silicon Gate
RAM stat.	Motorola MCM 10140	16 × 4 = 65	ECL	10 ns	bipolar
RAM stat.	Signetics 82 S 16	256 × 1	TTL	30 ns	bipolar
RAM stat.	RCA CD 4061	256 × 1	CMOS	300 ns	COS/MOS
RAM stat.	NS MM 1101	256 × 1	P-MOS	0,8 μs	Silicon Gate
RAM stat.	Intersil IM 6508	1024 × 1	CMOS	400 ns	Silicon Gate
RAM stat.	Intel 2102	1024 × 1	N-MOS	500 ns	Silicon Gate
RAM dyn.	Intel 2105	1024 × 1	N-MOS	95 ns	Silicon Gate
RAM dyn.	NS MM 4262	2048 × 1	P-MOS	470 ns	Silicon Gate
RAM dyn.	TI TMS 4030	4096 × 1	N-MOS	300 ns	–
SR stat.	Philips GZF 1106	4 × 64	CMOS	4 MHz	LOCMOS
SR stat.	TI TMS 3128	2 × 128	P-MOS	2,5 MHz	–
SR stat.	NS MM 5057	512	P-MOS	2,2 MHz	Silicon Gate
SR stat.	Intersil IM 7733	1024	N-MOS	1,5 MHz	–
SR dyn.	NS MM 4025	2 × 1024	P-MOS	6 MHz	Silicon Gate
SR dyn.	Intel 2405	2048	N-MOS	1 MHz	Silicon Gate

Das Programmieren von Mikrocomputern

Die Mikrocomputer haben in sehr kurzer Zeit eine grosse Verbreitung gefunden. Elektronikspezialisten jeder Stufe werden deshalb, oft recht unerwartet, mit Problemen der Computertechnik konfrontiert.
Im ersten Aufsatz dieser Broschüre (Seite 9) wird die neue Technik vorgestellt und werden die Probleme beleuchtet, die sich bei der Arbeit mit den Mikrocomputern ergeben können. Während die Schaltungstechnik für den Elektroniker keine allzu grossen Probleme aufwirft, ist ihm das Programmieren meistens vollkommen fremd (siehe Aufsatz Seite 17). Wie kann er sich raschmöglichst die nötigen Kenntnisse aneignen? Der vorliegende Artikel soll nicht nur eine Antwort auf diese Frage geben, sondern auch Gelegenheit bieten, durch sorgfältige Lektüre und das Studium der Beispiele sich im Selbststudium in dieses Gebiet einzuarbeiten.
Da im ersten Aufsatz der Rechner MCS-4 von Intel aus der grossen Zahl von Mikrocomputern als Beispiel ausgewählt wurde, wird auch in diesem Aufsatz auf diesen Rechner Bezug genommen. Die vermittelten Kenntnisse der Programmiertechnik sind aber vom Rechnertyp unabhängig.

1. Das Vorgehen zum Erlernen des Programmierens

1.1 Kurse

Programmierkurse werden von verschiedensten Organisationen angeboten. Meist handelt es sich um Kurse für sogenannt höhere Programmiersprachen (Cobol, Fortran, PL 1 usw.) wie sie in Anwendungen der kommerziellen und der administrativen Datenverarbeitung eingesetzt werden. «Rechnernahe» Programmiersprachen, wie sie beim Programmieren von Prozesssteuerungen eingesetzt werden, werden fast nur von den Herstellern der Rechner selbst unterrichtet. Nur wenige Kurse finden in der Schweiz statt, und oft sind die Kosten recht hoch.

1.2 Selbststudium

Es ist durchaus möglich, das Programmieren für Mikrocomputer im Selbststudium zu erlernen. Voraussetzung dafür ist ein gewisses Verständnis für den Aufbau von Computern und die Verarbeitung von Daten in Computern, wie sie in Abschnitt 2 und 3 dieses Aufsatzes vermittelt werden. Wenn man danach die Programmierung eines bestimmten Rechners erlernen will, studiert man zuerst das Instruktionsrepertoire, d.h. die verschiedenen Befehle des betreffenden Rechners, wie sie in Abschnitt 4 für den Rechner MCS-4 aufgeführt sind, und versucht die Programmbeispiele, wie sie fast immer in den Handbüchern der Computer zu finden sind, zu verstehen. In diesem Fall werden zwei Beispiele in den Abschnitten 5 und 6 eingehend erläutert. Dann stellt man sich selbst eine einfache Aufgabe und versucht, das entsprechende Programm zu schreiben. Wenn möglich sollte man dieses erste Programm und auch die paar nächstfolgenden mit einem auf dem betreffenden Rechner erfahrenen Programmierer besprechen. In Ermangelung eines solchen überarbeitet man die ersten Programme mehrmals und versucht dabei jedesmal, durch Verwendung geeigneterer Instruktionen, das Programm zu verbessern. Dieser Vorgang ist am ersten Beispiel verdeutlicht.
Gerade beim Programmieren gilt die alte Regel «Übung macht den Meister» ganz besonders. Ist aber erst einmal das Programmieren eines bestimmten Rechners erlernt, so ist das Umlernen auf den zweiten oder dritten Rechnertyp eine Frage von wenigen Tagen. Auch zum Umlernen empfiehlt es sich, das oben beschriebene Vorgehen anzuwenden.

2. Der Aufbau eines Computers

2.1 Allgemeines

Bild 1 zeigt den prinzipiellen Aufbau eines Computers:

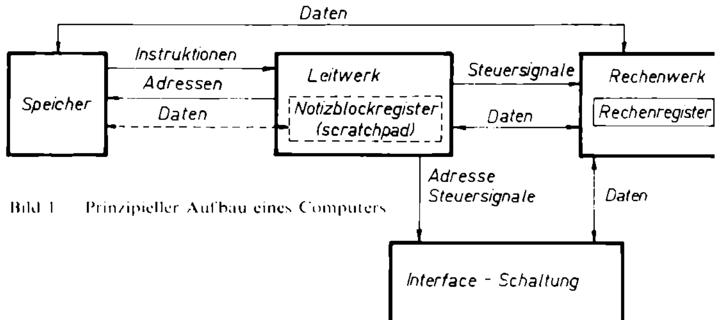


Bild 1 Prinzipieller Aufbau eines Computers

2.1.1 Das Rechenwerk

Hier werden Daten (z.B. Messwerte, Zahlen) miteinander verknüpft. Das Resultat von Operationen steht normalerweise im Rechenregister, auch Akkumulator genannt. Vom Rechenwerk hat der Computer auch seinen Namen: «Rechnen» heisst auf englisch «to compute».

2.1.2 Das Leitwerk

Das Leitwerk sorgt für einen geordneten Ablauf: Gemäss einem zum Leitwerk gehörenden Programmzähler wird dem Speicher eine Instruktion nach der anderen entnommen. Entsprechend jeder Instruktion werden die Datenwege zum und vom Rechenwerk freigegeben und die Operationen des Rechenwerks gesteuert. Um einige der, insbesondere bei Mikrocomputern oft recht umständlichen, Verschiebungen von Daten vom und zum Speicher zu vermeiden, ist das Leitwerk oft mit einer Anzahl von Notizblockregistern ausgerüstet, in denen eine beschränkte Anzahl Daten vorübergehend gespeichert werden können.

2.1.3 Der Speicher

Der Speicher enthält vor allem das Programm. Je nach der Art des Speichers können auch Daten gespeichert werden (Schreib-Lese-Speicher). Das Leitwerk sorgt für korrekte Adressierung und entsprechende Steuerbefehle (Schreib- oder Lesebefehl, Takt). Im Speicher sind Befehle oder Daten in rein binärer Form gespeichert.

2.1.4 Interface-Schaltungen

Über die Interface-Schaltungen werden Daten zwischen dem Rechner und den angeschlossenen Geräten ausgetauscht. Um zwischen den verschiedenen Geräten auswählen zu können, muss das Leitwerk eine Peripherieadresse abgeben.

2.2 Besonderheiten des Rechners MCS-4 von Intel

Es werden hier die Besonderheiten des Rechners nur so weit erläutert, wie sie für die folgenden Beispiele relevant sind. Für die genauere Beschreibung wird auf den Artikel auf Seite 9 und die Handbücher des Rechners verwiesen.

2.2.1 Das Rechenwerk

Das Rechenwerk verarbeitet parallel 4 Binärstellen (engl.: bits), d.h., es können z.B. 4 bit zu 4 bit addiert werden, wobei das Resultat wieder 4 bit umfasst, zuzüglich eines Überlaufbits (carry).

2.2.2 Das Leitwerk

Das Leitwerk des Rechners MCS-4 schliesst einen Notizblock mit 16 Registern ein, die einzeln oder paarweise gemäss Tabelle 1 adressiert werden. Jedes Register umfasst 4 Binärstellen. Sofern nicht ausdrücklich vom Rechenregister die Rede ist, sollen im folgenden unter dem Begriff «Register» die Register des Notizblocks verstanden werden.

Tabelle 1 Registerpaare

7P	14	15
6P	12	13
5P	10	11
4P	8	9
3P	6	7
2P	4	5
1P	2	3
0P	0	1

2.2.3 Der Speicher

Im Rechner MCS-4 werden Daten ausserhalb des Notizblockspeichers normalerweise getrennt vom Programm gespeichert, und zwar im Hauptdatenspeicher oder im Kontrollwortspeicher, die vor dem Zugriff durch getrennte Befehle adressiert werden müssen. Der separate Programmspeicher umfasst pro Adresse 8 Binärstellen. Der Programmspeicher ist normalerweise als Nur-Lesespeicher (ROM) ausgeführt. So geht das Programm auch bei Stromausfall nicht verloren. Sofern der Programmspeicher als Schreib-Lese-Speicher ausgeführt ist, können auch dort Daten gespeichert werden. Auch in diesem Fall erfolgt die Adressierung in einem getrennten Befehl.

2.2.4 Die Interface-Schaltungen

Daten werden immer nur zwischen dem Rechenregister und den Interface-Schaltungen ausgetauscht. Die Peripherieadresse muss in einem geradzahligen Register gespeichert sein und wird auf einen besonderen Befehl von dort ausgesendet. Es werden verschiedene Interfaces unterschieden, die bausteinmässig entweder mit dem Programm- oder dem Datenspeicher assoziiert sind. Die Ausgabekanäle sind normalerweise so ausgeführt, dass die Daten, bis zur nächsten Ausgabe über den gleichen Kanal, gespeichert bleiben.

3. Instruktionen und Daten

3.1 Befehlsfolge

Die einzelnen im Programm aufeinanderfolgenden Befehle sind im Speicher normalerweise unmittelbar nacheinander in Speicherzellen mit aufsteigender Adresse gespeichert. Eine Ausnahme bilden die Sprungbefehle, die zum Beispiel den Übergang zu einem anderen Programmteil erlauben.

3.2 Darstellungsart der Befehle

Im Speicher sind die Befehle in rein binärer Form gespeichert. Im Rechner MCS-4 sagt zum Beispiel die Bitfolge «1111 0010»: Addiere 1 zum Inhalt des Rechenregisters. Das Leitwerk interpretiert diesen binär verschlüsselten Befehl und beeinflusst das Rechenwerk entsprechend. Der menschliche Verstand hat dagegen mit dieser Schreibweise etwas Mühe. Deshalb werden für das Programmieren mnemotechnische, d.h. leicht zu erlernende Abkürzungen gebraucht, die fast immer auf der englischen Beschreibung des Befehls basieren. So lautet die Abkürzung für den oben beschriebenen Befehl «IAC» für «Increment ACcumulator». Neben der Abkürzung für den Befehlscode müssen im Programm oft ergänzende Angaben gemacht werden, so zum Beispiel, welches Register vom Befehl beeinflusst werden soll.

3.3 Binäre Zahlendarstellung

Daten werden vom Computer nur in binärer Form verarbeitet. Das Verständnis der binären Zahlendarstellung ist deshalb Voraussetzung für diese Art von Arbeit mit Computern.
Eine binäre Zahl besteht aus lauter Einern und Nullen. Wie im uns geläufigen Zehnersystem hat die rechte Stelle die tiefste, die linke die höchste Wertigkeit (Hunderter, Zehner, Einer) und der Wert einer Zahl ergibt sich aus der Summe aller mit ihrem Stellenwert multiplizierten Ziffern. Dazu ein Beispiel:

Stellenwert:	256	128	64	32	16	8	4	2	1
Beispiel:	1	0	1	1	0	0	0	1	0

Wert $= (1\cdot256)+(0\cdot128)+(1\cdot64)+(1\cdot32)+(0\cdot16)+(0\cdot8)+(0\cdot4)+(1\cdot2)+(0\cdot1) = 354$

Die Addition zweier binärer Zahlen wird wie im Zehnersystem durchgeführt: Man schreibt die beiden Summanden stellenrichtig untereinander und addiert die untereinanderstehenden Ziffern von rechts nach links. Dabei ist aber zu beachten, dass das binäre Zahlensystem nur die Ziffern 0 und 1 kennt. Ergibt die Addition zweier Ziffern mehr als 1, so ergibt sich ein Übertrag in die nächsthöhere Stelle, gleich wie bei der Addition von 5 und 7 im Zehnersystem.

Beispiel:

```
   011010
 + 111011
  1010101
```

4. Das Instruktionsrepertoire des Rechners MCS-4

4.1 Befehlsliste

Tabelle 2 zeigt die Befehle, wobei einige Einschränkungen und Ergänzungen, deren Erläuterung zu viel Raum beanspruchen würde, dem Handbuch des Rechners entnommen werden müssen. In der ersten Kolonne sind die mnemotechnischen Befehlscodes aufgeführt, wie sie für die Programmierung in Assemblersprache verwendet werden. Befehle, die in zwei Speicherworten gespeichert sind, sind mit * gekennzeichnet. Die zweite Kolonne zeigt die binären Befehlscodes, die sogenannte Maschinensprache. Sie gibt an, wie die Befehle im Programmspeicher enthalten sind. Damit man sich unter den mnemotechnischen Befehlscodes auch etwas vorstellen kann, sind in der dritten Kolonne die englischen Befehlsbeschreibungen angegeben, von denen die Abkürzungen abgeleitet

Tabelle 2 Befehlsliste des Rechners MCS-4

mnemo-techn. Befehl	binärer Befehl	englische Befehlsbeschreibung	Beschreibung der Operation	Bemerkungen zur Anwendung und Ergänzung der Befehlsbeschreibung
NOP	0000 0000	No OPeration	Keine Operation.	Wird u.a. für Zeitverzögerungen benützt.
LDM	1101 DDDD	Load Data from Memory	Die 4 in DDDD enthaltenen Datenbits werden vom Programmspeicher ins Rechenregister übertragen.	Benützt zur Initialisierung des Rechenregisters.
LD	1010 RRRR	Load Data	Der Inhalt des Registers RRRR wird ins Rechenregister kopiert.	
XCH	1011 RRRR	eXCHange	Austauschen der Inhalte von Register RRRR und des Rechenregisters.	Speichern des Inhaltes des Rechenregisters ohne Änderung des Rechenregisters nur mit Befehlsfolge XCH, LD möglich.
ADD	1000 RRRR	ADD register	Die Daten im Register RRRR werden zum Inhalt des Rechenregisters addiert, bzw. von ihm subtrahiert.	Wenn kein Übertrag zu berücksichtigen ist, vorher Übertragsbit löschen. Ergibt sich ein Übertrag, so ist das Übertragsbit nach ADD = 1, nach SUB = 0 bzw. umgekehrt, wenn sich kein Übertrag ergibt.
SUB	1001 RRRR	SUBtract register		
INC	0110 RRRR	INCrement register	Erhöhen des Inhalts von Register RRRR um 1.	Keine Beeinflussung des Übertragsbits.
ISZ*	0111 RRRR BBBB CCCC	Increment and Skip on Zero	Wie INC. Ist das Resultat = 0000, so wird mit dem folgenden Befehl weitergefahren, sonst zum Befehl an der Adresse BBBB CCCC im gleichen Sektor gesprungen.	Keine Beeinflussung des Übertragsbits. Anwendung in Schlaufen mit Kontrollzähler.
JUN*	0100 AAAA BBBB CCCC	Jump UNconditionally	Unbedingter Sprung zum Befehl an der Adresse AAAA BBBB CCCC.	
JCN*	0001 cccc BBBB CCCC	Jump CoNditionally	Bedingter Sprung innerhalb des gleichen Sektors zur Adresse BBBB CCCC, wenn die Bedingung cccc erfüllt ist, sonst Ausführung der nächsten Instruktion.	Bedingung cccc kann direkt im mnemotechnischen Befehlscode ausgedrückt werden (siehe Beispiele). Als Bedingungen cccc kommen «Rechenregister null», «Übertrag», «externes Signal TEST», Verneinung dieser Bedingungen, auch in Kombination in Frage (siehe dazu das Handbuch).
JIN	0011 PPP1	Jump INdirect	Unbedingter Sprung innerhalb des gleichen Sektors, nach der Adresse, die im Registerpaar PPP gespeichert ist.	Verwendung für Entscheidungstabellen.
FIM*	0010 PPP0 DDDD DDDD	Fetch IMmediate	Laden der Daten DDDD DDDD ins Registerpaar PPP.	Initialisieren von Registerpaaren.
FIN	0011 PPP0	Fetch INdirect	Laden der Daten aus dem Programmspeicher, die im gleichen Sektor an der Adresse gespeichert sind, die im Registerpaar 0 enthalten ist.	Laden von Daten aus Entscheidungsmatrizen oder Umcodierungstabellen.
SRC	0010 PPP1	Send Register Control	Adressieren von Interface-Schaltungen und Datenspeicher.	Siehe separate Erläuterung unter 4.2.
JMS*	0101 AAAA BBBB CCCC	JuMp to Subroutine	Sprung in ein Unterprogramm mit Speicherung der Rückkehradresse.	Maximal können 3 Rücksprungadressen gespeichert werden. Maximal dürfen also 3 Unterprogramme ineinander verschachtelt werden. Für die Rückkehr gilt immer die Rückkehradresse des jüngsten Einsprungs, für den noch keine Rückkehr erfolgte. Anwendung siehe Beispiel 2.
BBL	1100 DDDD	Branch Back and Link	Rücksprung aus einem Unterprogramm und gleichzeitig Laden des Rechenregisters mit den Daten DDDD.	
CLB	1111 0000	CLear Both	Löschen des Rechenregisters und des Übertragsbits.	
CLC	1111 0001	CLear Carry	Löschen des Übertragsbits.	
STC	1111 1010	SeT Carry	Setzen des Übertragsbits.	
IAC	1111 0010	Increment ACcumulator	Erhöhen des Inhalts des Rechenregisters um 1.	Das Übertragsbit wird gesetzt, wenn von 1111 auf 0000 erhöht wird, sonst wird es gelöscht.
DAC	1111 1000	Decrement ACcumulator	Vermindern des Inhalts des Rechenregisters um 1.	Das Übertragsbit wird beim Übergang von 0000 auf 1111 gelöscht, sonst immer gesetzt.
CMC	1111 0011	CoMplement Carry	Umkehren des Übertragsbits.	
CMA	1111 0100	CoMplement Accumulator	Inversion des Inhalts des Rechenregisters.	z.B. 1011 wird zu 0100. Übertragsbit unbeeinflusst.
TCC	1111 0111	Transmit and Clear Carry	Löschen des Rechenregisters. Kopieren des Übertragsbits in die tiefste Stelle des Rechenregisters und Löschen des Übertragsbits.	
TCS	1111 1001	Transfer Carry for Subtraction	Rechenregister wird dezimal = 10, wenn Übertrag gesetzt war, sonst = 9. Das Übertragsbit wird gelöscht.	Dezimale (BCD) Subtraktion.
DAA	1111 1011	Decimal Adjust Accumulator	Wenn das Rechenregister dezimal 10 oder mehr enthält oder das Übertragsbit gesetzt ist, wird zum Inhalt des Rechenregisters 6 addiert und das Übertragsbit gesetzt, sonst wie NOP.	Dezimale Addition und Subtraktion. Die Addition von 5(0101) und 7(0111) ergibt 1100, was keiner dezimalen Zahl entspricht. Die Addition von 6(0110) ergibt das korrekte Resultat 2(0010) und Übertrag zur nächsthöheren Stelle.
KBP	1111 1100	KeyBoard Process	Umwandlung des Inhalts des Rechenregisters gemäss Tabelle im Handbuch.	Verarbeitung der von Tastaturen eingelesenen Daten mit programmierter Doppelbetätigungssperre.
RAL	1111 0101	Rotate Accumulator Left	Schieben des Inhalts des Rechenregisters um 1 bit nach links bzw. rechts. Das zum Rechenregister hinausgeschobene Bit wird in das Übertragsbit kopiert und dessen vorheriger Zustand ins Rechenregister nachgeschoben.	RAL: [Über-trags-bit] ← [Rechenregister] ← (Diagramm) Für RAR werden die Pfeilrichtungen umgekehrt.
RAR	1111 0110	Rotate Accumulator Right		
DCL	1111 1101	Designate Command Line	Auswahl unter verschiedenen Gruppen von Datenspeichern.	Siehe Abschnitt 4.2 und Handbuch.

mnemo-techn. Befehl	binärer Befehl	englische Befehlsbeschreibung	Beschreibung der Operation		Bemerkungen zur Anwendung und Ergänzung der Befehlsbeschreibung
WRM	1110 0000	WRite into Memory	Abspeichern des Inhalts des Rechenregisters im Datenspeicher.	Adressierung gemäss letztem SRC-Befehl	
RDM	1110 1001	ReaD from Memory	Lesen von Daten aus dem Datenspeicher in das Rechenregister.		
ADM SBM	1110 1011 1110 1000	ADd Memory SuBtract Memory	Addition bzw. Subtraktion der Daten aus dem Datenspeicher zum bzw. vom Inhalt des Rechenregisters.		Beeinflussung des Übertragsbits wie bei den Instruktionen ADD und SUB.
WR0 WR1 WR2 WR3	1110 0100 1110 0101 1110 0110 1110 0111	WRite status character n (n = 0, 1, 2, 3)	Abspeichern des Inhalts des Rechenregisters in einem der Kontrollworte des Datenspeichers.		
RD0 RD1 RD2 RD3	1110 1100 1110 1101 1110 1110 1110 1111	ReaD status character n (n = 0, 1, 2, 3)	Lesen der Daten aus einem der Kontrollworte des Datenspeichers in das Rechenregisters.		
WPM	1110 0011	Write into Program Memory	Abspeichern des Inhalts des Rechenregisters im Programmspeicher. (Nur möglich, wenn Schreib-Lese-Speicher.)		Abwechslungsweise wird die linke oder die rechte 4-bit-Gruppe des Speicherplatzes beeinflusst. Details siehe Handbuch.
WMP	1110 0001	Write into Memory Port	Ausgabe des Inhalts des Rechenregisters über den adressierten Ausgabekanal, der mit Datenspeicher assoziiert ist.		
WRR	1110 0010	WRite into ROMport	Ausgabe des Inhalts des Rechenregisters über den adressierten Ausgabekanal, der mit Programmspeicher (ROM) assoziiert ist.		
RDR	1110 1010	ReaD from ROMport	Lesen der am adressierten Eingabekanal anliegenden Daten in das Rechenregister.		

sind. In den beiden letzten Kolonnen sind sodann die Befehle auf deutsch beschrieben und, wo nötig, erläutert.
Bei allen Schreib- oder Lesebefehlen bleiben die Daten an ihrem Herkunftsort unverändert.
Unter dem Begriff «Speichersektor» wird der Bereich des Speichers verstanden, für dessen Adressierung gegenüber der Adresse des betreffenden Befehls die 4 signifikantesten Bits unverändert bleiben können. Ein Sektor umfasst also 256 Speicherzellen.
Die Ausführungszeit für eine Instruktion beträgt etwas über 10 μs, im Falle von 2-Wort-Befehlen das doppelte.

4.2 Adressieren von Datenspeicher, Ein- und Ausgabekanälen

Die Adressierung erfolgt durch die Ausgabe des Inhaltes eines Registerpaares mittels eines SRC-Befehls. Der zuletzt ausgeführte SRC-Befehl ist immer für die Adressierung aller so beeinflussbaren Elemente massgebend.

4.2.1 Adressierung des Datenspeichers

Der Datenspeicher ist in Kolonnen aufgeteilt. Je 16 Kolonnen sind zu einer Gruppe zusammengefasst. Soll eine andere als die laufende Gruppe adressiert werden, so ist zusätzlich zum SRC-Befehl noch ein DCL-Befehl auszugeben, mit dem unter maximal 7 Gruppen ausgewählt werden kann. Innerhalb einer Gruppe sind die Kolonnen von 0 bis 15 numeriert. Für die Wahl der Kolonne ist der Inhalt des gerade numerierten Registers, das mit dem SRC-Befehl ausgesendet wird, verantwortlich. Jede Kolonne besteht aus 4 Kontrollworten (0...3) und 16 Hauptspeicherworten (0...15).
Zu den Kontrollworten kann nach der Wahl einer Kolonne mit den entsprechenden Befehlen immer direkt zugegriffen werden, wogegen jedes Hauptspeicherwort mit einem SRC-Befehl gesondert adressiert werden muss. Für die Auswahl zwischen den Hauptspeicherworten ist der Inhalt des ungerade numerierten mit dem SRC-Befehl ausgesendeten Registers massgebend. Alle Datenworte umfassen 4 Bits.

4.2.2 Adressierung von Daten im Programmspeicher

Beim Schreiben von Daten in den Programmspeicher mit dem WPM-Befehl bestimmt der gesamte Inhalt des mit dem letzten SRC-Befehl ausgesendeten Registerpaares die Speicheradresse innerhalb eines Sektors. Der Sektor muss über einen Ausgabekanal, von der übrigen Speicheradressierung also vollständig getrennt, bestimmt werden.

4.2.3 Adressierung von mit dem Datenspeicher assoziierten Ausgabekanälen

Je 4 Kolonnen des Datenspeichers ist ein Ausgabekanal zugeordnet. Die Adressierung erfolgt mit den beiden vordersten Binärstellen des gerade numerierten Registers, das mit dem letzten SRC-Befehl ausgegeben wurde. Pro Speichergruppe sind also 4 solche Ausgabekanäle adressierbar.

4.2.4 Adressierung von mit dem Programmspeicher assoziierten Ein- und Ausgabekanälen

Total können je 16 Ein- und Ausgabekanäle adressiert werden. Der gesamte Inhalt des gerade numerierten Registers, das mit dem letzten SRC-Befehl ausgegeben wurde, ist für die Adressierung massgebend.

5. Programmbeispiel 1

5.1 Aufgabe

Das einfache Schema gemäss Bild 2a soll mit einem MCS-4 Computer realisiert werden. Es ergibt sich das Schema gemäss Bild 2b.

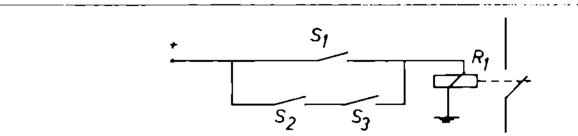


Bild 2a Relais-Schaltung

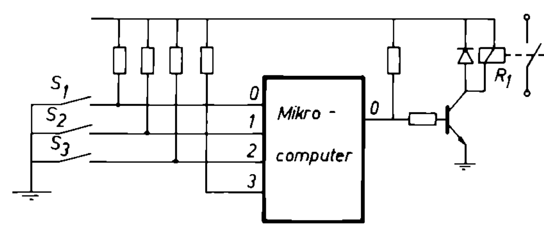


Bild 2b Realisierung mit Mikrocomputer

5.2 Erste Lösung

Diese Lösung ist bewusst nicht optimal, sondern so, wie sie einem Programmieranfänger als erster Versuch gelingen könnte. Zum Programm, das in Tabelle 3 gezeigt ist, folgender Kommentar:

Tabelle 3 Programm 1

```
 0:    LDM 0
 1:    XCH 0
 2:    SRC 0
 3:    LDM 3
 4:    XCH 2
 5:    RDR
 6:    CMA
 7:    CLC
 8:    RAR
 9:    JOC 18
11:    SUB 2
12:    JAZ 18
14:    LDM 0
15:    WMP
16:    JUN 0
18:    LDM 1
19:    JUN 15
21:$
```

Die Zahlen entlang dem linken Rand geben die Adresse des Speicherplatzes an, an der sich der auf der gleichen Zeile aufgeführte Befehl befindet.

Mit der ersten Instruktion LDM wird das Rechenregister gelöscht. Der Wert 0 wird mit der Instruktion XCH vom Rechenregister in das Register 0 übertragen, von wo aus der Wert mit dem SRC-Befehl zur Auswahl der anzusteuernden Interface-Schaltungen ausgesendet wird. Da im ganzen Programm nur Interface-Schaltungen mit der Adresse 0 angesprochen werden, bleibt dies der einzige SRC-Befehl. Wie vorher das Register 0 wird mit den nächsten beiden Instruktionen das Register 2 auf einen festen Wert initialisiert, nämlich auf 3. Mit RDR werden nun die Schalter abgefragt. Danach steht in jedem Bit des Rechenregisters, das einem Schalter entspricht, eine 0, wenn der Schalter geschlossen war und sonst eine 1. Damit einem geschlossenen Schalter eine 1 entspricht und einem offenen eine 0, wird der Inhalt des Rechenregisters mit CMA komplementiert. Damit beim folgenden Schiebebefehl sicher eine 0 in das Rechenregister eingeschoben wird, wird das Übertragsbit mit CLC gelöscht. Mit dem Schiebebefehl RAR wird das dem Schalter S1 entsprechende Bit in das Übertragsbit geschoben. War der Schalter geschlossen, ist dieses Bit also jetzt gesetzt, und mit dem bedingten Sprung JOC (Jump On Carry) wird zur Adresse 18 gesprungen. War der Schalter S1 aber offen, so müssen noch die Stellungen der beiden anderen Schalter geprüft werden. Die diesen Schaltern entsprechenden Bits stehen nach dem Schiebebefehl ganz rechts im Rechenregister. Sind beide Schalter geschlossen, so enthält das Rechenregister jetzt den Wert 3. Dies wird geprüft, indem man vom Inhalt des Rechenregisters mit dem Befehl SUB den Inhalt des Registers 2, der auf 3 initialisiert worden war, abzieht. Waren beide Schalter geschlossen, so enthält das Rechenregister jetzt 0. In diesem Fall wird mit dem bedingten Sprung JAZ (Jump if Accumulator Zero) ebenfalls zur Adresse 18 gesprungen. Andernfalls wird mit LDM das Rechenregister wieder gelöscht und dieser Wert mit WMP zur Ansteuerung des Relais über den Ausgabekanal ausgegeben. Dank dem Sprung mit JUN auf Adresse 0 wird das ganze Programm zyklisch wiederholt. Im Falle, dass eine der beiden Bedingungen für das Anziehen des Relais erfüllt war, wurde auf Adresse 18 gesprungen. Dort wird mit LDM das tiefste Bit des Rechenregisters gesetzt. Der Sprung mit JUN führt zum bereits besprochenen Ausgabebefehl.

5.3 Kritik der ersten Lösung

Anstelle des Registers 2 hätte ebensogut das Register 1 zur Speicherung des Wertes 3 benutzt werden können. Die Initialisierung des Registers 0 auf den Wert 0 und des Registers 1 auf den Wert 3 hätte dann mit dem einzigen Befehl «FIM 0P 3» vorgenommen werden können.

Es ist nicht nötig, dass das Programm alle Instruktionen, die zur Initialisierung benötigt werden, bei jedem Umlauf erneut ausführt. Ein Rücksprung zum Einlesebefehl RDR anstatt zur Adresse 0 erhöht die Umlaufgeschwindigkeit des Programmes.

5.4 Zweite Lösung

Das Programm, wie es der gleiche Programmierer schon nach einigen Stunden Übung schreiben könnte, ist in Tabelle 4 dargestellt. Gegenüber der ersten Lösung weist es folgende Verbesserungen auf:

Das Programm ist mit Kommentar versehen, so dass der gleiche Programmierer auch nach ein paar Tagen, wenn er das Programm wieder zur Hand nimmt, sofort wieder weiss, worum es sich handelt. Kommentare sind die Texte, die hinter « » stehen. Sie haben natürlich keinen Einfluss auf den Ablauf des Programms bei der Ausführung und belegen auch keinen Speicherplatz, sondern erscheinen nur auf der Programmliste.

Anstelle der sogenannt absoluten Adressierung ist die symbolische Adressierung verwendet worden. So muss der Programmierer nicht mehr selbst abzählen, an welche Speicheradresse ein bestimmter Befehl zu stehen kommt, sondern er gibt ihm einen Namen wie «EIN» oder «AUS» und bezieht sich bei Sprungbefehlen auf diese Namen.

Das Programm kommt mit der unter 5.3 erwähnten kurzen Initialisierung aus. Genau genommen muss sogar nur das Register 0 initialisiert werden, was sowohl mit FIM wie auch mit der Befehlsfolge LDM, XCH gemacht werden kann. Beide Arten benötigen gleich viel Speicherplatz und Ausführungszeit. Nachdem das Bit für den Schalter S1 in das Übertragungsbit geschoben wurde, werden auch die Bits der anderen Schalter dorthin geschoben. Ist

Schalter S2 offen, so ist das Übertragsbit = 0, und es wird mit JNC (Jump if No Carry) nach AUS gesprungen. War S2 geschlossen, so wird AUS ohne Sprung erreicht, und das Übertragsbit entspricht der Stellung von S3. Wie also auch immer AUS erreicht wurde, entspricht die Stellung des Übertragsbits der Sollstellung des anzusteuernden Relais. Das Übertragsbit wird mit TCC in das Rechenregister übertragen. Ebensogut hätte dazu RAL verwendet werden können. In diesem Fall wird das Relais-Steuerbit mit WMP ausgegeben, darauf wird zum Einlesebefehl zurückgesprungen.

Tabelle 4 Verbessertes Programm 1

```
 0:/BEISPIEL 1, VERBESSERTE LOESUNG
 0:     FIM 0 0
 2:     SRC 0
 3:EIN RDR      / EINLESEN DER SCHALTERSTELLUNGEN
 4:     CMA      / SCHALTER ZU = 1
 5:     RAR
 6:     JOC AUS
 8:     RAR
 9:     JNC AUS
11:     RAR
12:AUS TCC      / AUSGABE DES RELAIS-SCHALTBEFEHLS
13:     WMP
14:     JUN EIN
16:$
```

6. Programmbeispiel 2

6.1 Aufgabenstellung

Von zwei Messgeräten wird je ein Messwert abgegeben, der 8 Binärstellen umfasst. Ein Taktgeber signalisiert periodisch, zum Beispiel alle 2 Sekunden, die Bereitstellung neuer Messwerte. Diese Messwerte sollen zusammengezählt werden. Ist die Summe kleiner als 300, so soll ein Schalter geschlossen, ist sie grösser als 400, so soll er geöffnet werden. Die Summe soll ausserdem dezimal angezeigt werden.

Eine solche Summenbildung mit Auslösung durch ein Taktsignal, der Vergleich mit Grenzwerten und selbst die Zerlegung in anzuzeigende Ziffern sind schaltungsmässig lösbar. Trotzdem ist dies bereits ein relativ typisches Beispiel für die Anwendung eines Mikrocomputers. Würde statt der einfachen Summenbildung eine komplizierte Rechenoperation verlangt, so wäre der Einsatz eines Mikrocomputers bald unumgänglich.

Die Konfiguration der Ein- und Ausgabekanäle wird für das Programmbeispiel wie folgt festgelegt:

Eingänge: Kanal 0: Auslösesignal
Kanal 1 und 2: Messwert 1
Kanal 3 und 4: Messwert 2

Auf dem tiefer numerierten Kanal werden jeweils die weniger signifikanten Teile der beiden Messwerte eingelesen.

Ausgänge: Kanal 0: Ziffernwert für die Anzeige
Kanal 1: Bit 0: Schalteransteuerung
Bit 1: Speicherbefehl tiefste Ziffer
Bit 2: Speicherbefehl mittlere Ziffer
Bit 3: Speicherbefehl höchste Ziffer

Die Ausgaben erfolgen über die mit dem Programmspeicher assoziierten Ausgabekanäle.

6.2 Das Programmbeispiel

(Tabelle 5)

6.2.1 Das Warten auf den Taktgeber

Mit den drei ersten Befehlen wird wie im Beispiel 1 das Register 0 auf 0 initialisiert und der Eingabekanal 0 angewählt. Mit dem

Tabelle 5 Programm 2

```
  0:/ MUSTERPROGRAMM MCS 4, KA 10.12.74
  0:A00 LDM 0
  1:    XCH 0
  2:    SRC 0
  3:A05 RDR
  4:    JAZ A05
  6:    INC 0
  7:    SRC 0
  8:    RDR
  9:    XCH 3
 10:    INC 0
 11:    SRC 0
 12:    RDR
 13:    XCH 2
 14:    INC 0
 15:    SRC 0
 16:    RDR
 17:    CLC
 18:    ADD 3
 19:    XCH 3
 20:    INC 0
 21:    SRC 0
 22:    RDR
 23:    ADD 2
 24:    XCH 2
 25:    TCC
 26:    XCH 1
 27:    LDM 1
 28:    XCH 0
 29:    SRC 0
 30:    FIM 2P 1
 32:    FIM 3P 44
 34:    JMS TEST
 36:    LDM 1
 37:    JOC A10
 39:    FIM 3P 144
 41:    JMS TEST
 43:    JOC A15
 45:/
 45:A10 WRR
 46:    XCH 8
 47:A15 FIM 5P 50
 49:    JMS DIV
 51:    LDM 8
 52:    JMS ANZ
 54:    FIM 5P 80
 56:    JMS DIV
 58:    LDM 4
 59:    JMS ANZ
 61:    LD 1
 62:    XCH 6
 63:    LDM 2
 64:    JMS ANZ
 66:    JUN A00
 68:/
 68:/ UNTERPROGRAMME
 68:/
 68:TEST CLC
 69:    LD 7
 70:    SUB 3
 71:    CMC
 72:    LD 6
 73:    SUB 2
 74:    CMC
 75:    LD 5
 76:    SUB 1
 77:    BBL 0
 78:/
 78:DIV FIM 3P 12
 80:D10 CLC
 81:    LD 2
 82:    SUB 11
 83:    XCH 5
 84:    CMC
 85:    LD 1
 86:    SUB 10
 87:    XCH 4
 88:    LD 6
 89:    RAL
 90:    XCH 6
 91:    LD 6
 92:    RAR
 93:    JNC D20
 95:    LD 5
 96:    XCH 2
 97:    LD 4
 98:    XCH 1
 99:D20 CLC
100:    LD 3
101:    RAL
102:    XCH 3
103:    LD 2
104:    RAL
105:    XCH 2
106:    LD 1
107:    RAL
108:    XCH 1
109:    ISZ 7 D10
111:    BBL 0
112:/
112:ANZ XCH 7
113:    LDM 0
114:    XCH 0
115:    SRC 0
116:    LD 6
117:    WRR
118:    INC 0
119:    SRC 0
120:    CLC
121:    LD 8
122:    ADD 7
123:    WRR
124:    SUB 7
125:    WRR
126:    BBL 0
127:$
```

Befehl RDR werden dann die Daten von diesem Kanal eingelesen. Sofern die anderen drei Leitungen dieses Kanals richtig angeschlossen sind, wird das Rechenregister nach Ausführung dieser Instruktion immer 0 enthalten, solange der Taktimpuls nicht anliegt. Der Taktimpuls muss übrigens etwa 40 μs lang sein, damit er sicher eingelesen wird. Solange der Takt nicht anliegt, wird mit dem bedingten Sprung JAZ (Jump if Accumulator Zero) immer nach A05 zurückgesprungen, und die Daten werden von Kanal 0 erneut eingelesen. Erst wenn der Taktimpuls anliegt, fährt das Programm mit dem nächsten Befehl fort.

6.2.2 Das Einlesen der Messwerte und Summenbildung

Mit INC wird der Inhalt des Registers 0 um 1 erhöht, bevor es mit SRC ausgesendet wird, um Eingabekanal 1 anzuwählen. Die mit RDR über den gewählten Kanal eingelesenen Daten werden mit XCH im Register 3 abgespeichert. In der gleichen Weise wird mit den nächsten 4 Befehlen der zweite Teil des ersten Messwertes über Kanal 2 eingelesen und in Register 2 gespeichert. Auch die an Kanal 3 anliegenden Daten werden auf gleiche Art in das Rechenregister gelesen, jedoch nicht wie die Daten des ersten Messwertes abgespeichert. Für die folgende Summenbildung wird zuerst das Übertragsbit gelöscht. Dann wird zu dem im Rechenregister stehenden weniger signifikanten Teil des zweiten Messwertes aus Register 3 der entsprechende Teil des ersten Messwertes mit ADD addiert. Das Resultat wird mit XCH wieder in Register 3 abgespeichert. Wie alle Daten vorher wird auch der zweite Teil des zweiten Messwertes über Kanal 4 in das Rechenregister eingelesen. Im Übertragsbit ist immer noch gespeichert, ob sich bei der Addition der weniger signifikanten Teile der Messwerte ein Übertrag ergeben hat. Das Bit muss unverändert für die folgende Addition berücksichtigt werden. Aus Register 2 werden die vorher eingelesenen Daten des ersten Messwertes zu den eben eingelesenen Daten addiert, und mit XCH wird das Resultat im Register 2 versorgt. Auch aus dieser zweiten Addition kann sich ein Übertrag ergeben haben, der mit TCC in das Rechenregister gebracht und von dort mit XCH in Register 1 gespeichert wird. Anders als im Beispiel 1 könnte die Instruktion RAL nicht anstelle von TCC verwendet werden, da der Rest des Rechenregisters gelöscht werden muss.

6.2.3 Vergleich mit den Grenzwerten und Ansteuern des Schalters

Zuerst wird, wie in beiden Beispielen bereits mehrfach, das Register 0 so initialisiert, dass mit SRC Kanal 1 gewählt wird. Dann wird der erste Grenzwert in die Register 5, 6 und 7 geladen. Der Wert (300) muss dazu zerlegt werden, da die grösste Zahl, die mit 8 Binärstellen dargestellt werden kann, 255 ist. Der Befehl FIM 2P 1 initialisiert das Registerpaar 2P auf den Wert 1, also Register 4 auf 0 und Register 5 auf 1. Diese 1 steht für 256. Die Differenz zu 300 beträgt 44. Auf diesen Wert werden, ebenfalls mit FIM, die Register 6 und 7 initialisiert. Das Register 6 wird 0010, entsprechend 32, und das Register 1 wird 1100, entsprechend 12, enthalten. 256 + 32 + 12 = 300! Mit dem Befehl JMS wird das Unterprogramm TEST gerufen, d.h. es wird so zur mit TEST markierten Instruktion gesprungen, dass später zu der unmittelbar auf den Aufruf des Unterprogramms folgenden Instruktion zurückgesprungen werden kann. Das Unterprogramm wird weiter unten besprochen. Vorläufig ist nur wichtig, dass bei der Rückkehr das Übertragsbit gesetzt ist, wenn die Summe der Messwerte kleiner ist als der Referenzwert.

Nach der Rückkehr aus dem Unterprogramm wird zuerst das Rechenregister so initialisiert, dass seine Ausgabe an den Ausgabekanal 1 das Schliessen des Schalters bewirken würde. Mit dem bereits bekannten Befehl JOC wird danach das Übertragsbit geprüft und nach A10 gesprungen, falls es gesetzt ist. Der immer noch im Register 5 geladene Wert 1 (= 256) ist auch für den zweiten Grenzwert gültig. Mit FIM wird das Registerpaar 3P auf 144 initialisiert, entsprechend binär 1001 0000. Wieder wird mit JMS das Unterprogramm TEST gerufen, das die Summe der Messwerte mit dem Referenzwert vergleicht. Ohne weiteres kann hier der Sinn dieser Unterprogramme erkannt werden: Wäre der Vergleich mit einem Grenzwert jedesmal wo erforderlich mit allen nötigen Instruktionen ausprogrammiert worden, so würde dafür doppelt so viel Speicherplatz benötigt als bei der Programmierung als Unterprogramm. Das gleiche Unterprogramm kann beliebig oft gerufen werden.

Nach der zweiten Rückkehr aus dem Unterprogramm TEST ist das Übertragsbit gesetzt, wenn die Summe der Messwerte kleiner ist als der Referenzwert, wenn also die Schalterstellung nicht verändert werden muss. Deshalb wird in diesem Fall die Ansteuerung des Schalters mit JOC übersprungen. Nach der Rückkehr aus dem Unterprogramm enthält das Rechenregister 0 und kann somit direkt zur Ansteuerung des Schalters verwendet werden. Je nach Inhalt des Rechenregisters wird bei A10 der Schalter geöffnet oder geschlossen, wenn die Daten mit WRR über den früher angewählten Ausgabekanal 1 abgegeben werden.

6.2.4 Zerlegung des Binärwertes in 3 Ziffern

Die Summe der beiden Messwerte kann nie grösser als 510 sein. Zur Zerlegung wird die Summe zuerst durch 100, der Rest dann durch 10 dividiert. Der Rest der zweiten Division ist die letzte Ziffer. Sowohl für die Division wie auch für die Ansteuerung der Anzeigen werden Unterprogramme benützt.

Die zu dividierende Summe steht in den Registern 1, 2 und 3. Der Divisor wird in Registerpaar 5P mit FIM initialisiert. Um das Unterprogramm DIV optimal gestalten zu können, wird der Divisor nur auf seinen halben Wert (50) initialisiert. Das Resultat des Unterprogramms DIV, ein Wert zwischen 0 und 5, steht in Register 6 und wird direkt an das Unterprogramm ANZ weitergegeben. Im Rechenregister wird mit LDM der Einleseimpuls für die betreffende Ziffer initialisiert. Der aus der ersten Division verbleibende Rest steht nun in den Registern 1 und 2. Im nicht mehr belegten Register 3 ist 0 eingeschoben worden. Der Divisor wird im Register 10 wiederum auf seinen halben Wert initialisiert (80 = binär 0101 0000). Register 10 enthält also den Wert 5. Da der Dividend der zweiten Division gegenüber demjenigen der ersten um 4 Binärstellen nach links verschoben ist, muss auch der Divisor entsprechend initialisiert werden.

Wie nach der ersten Division steht das Resultat in Register 6 und kann direkt weitergegeben werden, während der Einleseimpuls für die mittlere Ziffer im Rechenregister initialisiert werden muss. Der Rest der zweiten Division steht im Register 1. Bevor das Unterprogramm ANZ gerufen werden kann, muss er mit LD und XCH ins Register 6 gebracht werden. Auch hier muss vor dem Aufruf des Unterprogramms ANZ der Ziffer-Einleseimpuls ins Rechenregister geladen werden.

Nach der Anzeige der letzten Ziffer kann an den Anfang des Programms zurückgesprungen werden.

6.2.5 Das Unterprogramm TEST

Vor der Subtraktion der am wenigsten signifikanten Teile der

beiden Vergleichswerte muss das Übertragsbit gelöscht werden. Nun wird vom wenigst signifikanten Teil zum signifikantesten jeweils der entsprechende Teil des Referenzwertes mit LD ins Rechenregister geladen und mit SUB der entsprechende Teil der Messwertsumme abgezogen. Das Resultat muss nicht gespeichert werden. Zwischen den Subtraktionen muss der Wert des Übertragsbits umgekehrt werden, da dieses Bit nach einer Subtraktion gesetzt ist, wenn sich *kein* Übertrag ergeben hat. Nach der letzten Subtraktion kann direkt ins Hauptprogramm zurückgesprungen werden. Das Übertragsbit ist beim Rücksprung gesetzt, wenn der subtrahierte Wert kleiner ist als der Referenzwert. Beim Rücksprung mit BBL wird das Rechenregister immer mit dem Wert 0 geladen.

6.2.6 Das Unterprogramm DIV

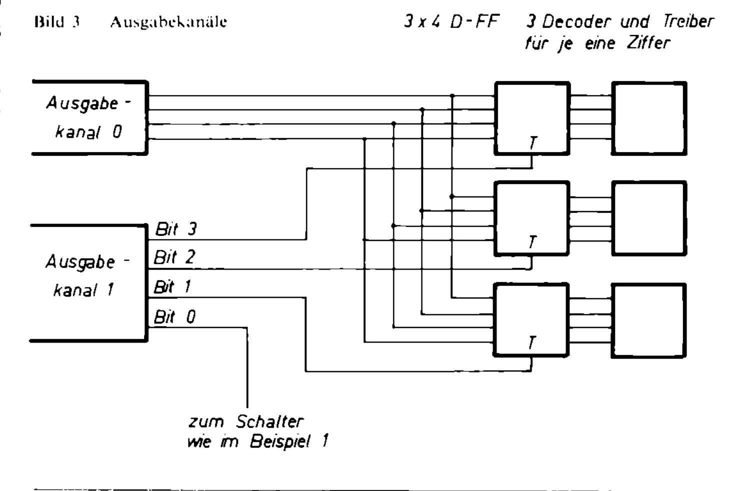
Gleich am Anfang wird mit FIM das Register 6 gelöscht und das Register 7 auf 12 initialisiert. Register 6 dient als Resultatspeicher, Register 7 als Zähler für die Division. Der Wert 12 dieses Zählers entspricht 4 Durchläufen (= Anzahl Binärstellen des Resultats) der nachfolgenden Programmschlaufe, da der Inhalt des Registers 4mal erhöht werden kann, bevor es überläuft.
Wie im Unterprogramm TEST wird vor der ersten Subtraktion das Übertragsbit mit CLC gelöscht. Die Subtraktion läuft hier nur über 2mal 4 bit, und das Resultat wird in den Registern 4 und 5 gespeichert. War der Dividend grösser als der Divisor, so ist nach der Subtraktion der zweiten 4-bit-Gruppe das Übertragsbit gesetzt und kann als erstes Bit des Resultates mit RAL in das Resultatregister 6 eingeschoben werden. Dass beim Einschieben die bisher in Register 6 enthaltenen Binärstellen um eine Stelle nach links geschoben wurden, hat erst beim zweiten und den weiteren Durchläufen durch die Subtraktionsschlaufe seine Bedeutung. Mit LD und RAR wird das Übertragsbit wieder in den Zustand vor dem Einschieben gebracht. War der Divisor grösser, so war die Subtraktion erfolgreich, und der in den Registern 4 und 5 gespeicherte Wert ist der Rest, der für die weitere Division massgebend ist. Der Inhalt dieser Register wird deshalb in die Register 1 und 2 übertragen. War der Dividend kleiner als der Divisor, so ist das Übertragungsbit gelöscht, und das Programm springt mit JNC nach D20.
Mit CLC wird das Bit, das in den Dividenden geschoben wird, gelöscht. Der Inhalt aller drei Register, die den Dividenden enthalten, wird sodann um eine Stelle nach links geschoben. Dieser gesamte Vorgang der Subtraktion und des Verschiebens des verbleibenden Dividenden wird 4mal wiederholt, nämlich so lange, bis bei ISZ das Register 7 von 1111 nach 0000 überläuft. Dabei sind also 4 Binärstellen in das Resultatregister 6 eingeschoben worden, und der Rest ist gegenüber dem ursprünglichen Dividenden um 4 Stellen nach links geschoben.
Am besten wird das Unterprogramm DIV dann verstanden, wenn es mit Papier und Bleistift an einem Beispiel durchexerziert wird.

6.2.7 Das Unterprogramm ANZ

Das im Rechenregister enthaltene Bit, das als Einleseimpuls für die anzuzeigende Ziffer an den Kanal 1 abgegeben werden muss, wird vorläufig mit XCH in Register 7 gespeichert. In der bereits bekannten Weise wird Register 0 so initialisiert, dass mit SRC der Kanal 0 für die Ausgabe der Ziffer angewählt wird. Der anzuzeigende Wert wird aus Register 6 mit LD ins Rechenregister geladen und von dort mit WRR an den gewählten Kanal abgegeben. Das Register 0 wird dann so erhöht, dass bei seiner Abgabe mit SRC der Kanal 1 angewählt wird. Da der Schalter und die Ziffer-Einlesebefehle über den gleichen Kanal abgegeben werden (ein nicht sehr empfehlenswertes Konzept), muss der aktuelle Schalterzustand mit dem Einlesebefehl kombiniert werden: Zuerst wird das Übertragsbit gelöscht und dann werden die in den Registern 7 und 8 gespeicherten Werte addiert. Die so kombinierten Werte werden mit WRR an den gewählten Kanal abgegeben. Der Einlesebefehl soll aber auch wieder gelöscht werden. Ein Übertrag hat sich aus der Addition sicher nicht ergeben, so dass der Inhalt von Register 7 direkt wieder subtrahiert werden kann. Mit WRR wird also nur noch der Schalter-Sollzustand abgegeben. Es wird nun offensichtlich, dass der Schalter im Hauptprogramm gar nicht hätte angesteuert werden müssen, da er zusammen mit der Ansteuerung der Ziffernanzeige sowieso richtig gestellt wird. Im Hauptprogramm könnten dadurch 4 Befehle eingespart werden.

Zum besseren Verständnis ist in Bild 3 die Schaltung der Ausgabekanäle dargestellt.

Bild 3 Ausgabekanäle



7. Das Assemblerprogramm

Wie entstehen aus den handgeschriebenen Programmen die durch den Computer ausführbaren, im Speicher des Computers enthaltenen Programme?
Vom Hersteller eines Rechners wird praktisch immer ein Assemblerprogramm geliefert. Dieses verarbeitet das meistens auf einem Lochstreifen abgelochte «Quellprogramm», wie es oben besprochen wurde. Es übersetzt dabei die mnemotechnischen Befehlscodes in die binären Maschinenbefehle, ordnet jedem Befehl die nötigen Speicherplätze zu und übersetzt vor allem die symbolischen Adressen in Speicheradressen, wie sie in der ersten Version des ersten Beispiels verwendet wurden. Selbstverständlich müssen auch alle dezimalen Angaben in Binärwerte umgewandelt werden.
Die Übersetzung des «Quellprogramms» des Beispiels 2 ist in Tabelle 6 gezeigt. Jedes Programmwort von 8 bit ist zwischen den Buchstaben B und F mit P und N dargestellt, wobei P für 0 und N für 1 steht.
Hätte das Programm ungültige Befehlscodes, doppelt definierte Namen, ungültige Registercodes usw. enthalten, so wären solche Fehler durch das Assemblerprogramm markiert worden.

Tabelle 6 Programm 2 in der Maschinensprache

```
  0:   0:BNNPNPPPPF   1:BNPNNPPPPF   2:BPPNPPPPNF   3:BNNNPNPNPF
  4:BPPPNPNPPF BPPPPPPNNF   6:BPNNPPPPPF   7:BPPNPPPPNF
  8:BNNNPNPNPF   9:BNPNNPPNNF  10:BPNNPPPPPF  11:BPPNPPPPNF
 12:BNNNPNPNPF  13:BNPNNPPNPF  14:BPNNPPPPPF  15:BPPNPPPPNF
 16:BNNNPNPNPF  17:BNNNNPPPNF  18:BNPPPPPNNF  19:BNPNNPPNNF
 20:BPNNPPPPPF  21:BPPNPPPPNF  22:BNNNPNPNPF  23:BNPPPPPNPF
 24:BNPNNPPNPF  25:BNNNNPNNNF  26:BNPNNPPPNF  27:BNNPNPPPNF
 28:BNPNNPPPPF  29:BPPNPPPPNF  30:BPPNPPNPPF BPPPPPPPNF
 32:BPPNPPNNPF BPPNPNNPPF  34:BPNPNPPPPF BPNPPPNPPF
 36:BNNPNPPPNF  37:BPPPNPPNPF BPPNPNNPNF  39:BPPNPPNNPF
BNPPNPPPPF  41:BPNPNPPPPF BPNPPPNPPF  43:BPPPNPPNPF
BPPNPNNNNF  45:  45:BNNNPPPNPF  46:BNPNNNPPPF  47:BPPNPNPNPF
BPPNNPPNPF  49:BPNPNPPPPF BPNPPNNNPF  51:BNNPNNPPPF
 52:BPNPNPPPPF BPNNNPPPPF  54:BPPNPNPNPF BPNPNPPPPF
 56:BPNPNPPPPF BPNPPNNNPF  58:BNNPNPNPPF  59:BPNPNPPPPF
BPNNNPPPPF  61:BNPNPPPPNF  62:BNPNNPNNPF  63:BNNPNPPNPF
 64:BPNPNPPPPF BPNNNPPPPF  66:BPNPPPPPPF BPPPPPPPPF
 68:  68:  68:  68:BNNNNPPPNF  69:BNPNPPNNNF  70:BNPPNPPNNF  71:BNNNNPPNNF
 72:BNPNPPNNPF  73:BNPPNPPNPF  74:BNNNNPPNNF  75:BNPNPPNPNF
 76:BNPPNPPPNF  77:BNNPPPPPPF  78:  78:BPPNPPNNPF BPPPPNNPPF
 80:BNNNNPPPNF  81:BNPNPPPNPF  82:BNPPNNPNNF  83:BNPNNPNPNF
 84:BNNNNPPNNF  85:BNPNPPPPNF  86:BNPPNNPNPF  87:BNPNNPNPPF
 88:BNPNPPNNPF  89:BNNNNPNPNF  90:BNPNNPNNPF  91:BNPNPPNNPF
 92:BNNNNPNNPF  93:BPPPNNPNPF BPNNPPPNNF  95:BNPNPPNPNF
 96:BNPNNPPNPF  97:BNPNPPNPPF  98:BNPNNPPPNF  99:BNNNNPPPNF
100:BNPNPPPNNF 101:BNNNNPNPNF 102:BNPNNPPNNF 103:BNPNPPPNPF
104:BNNNNPNPNF 105:BNPNNPPNPF 106:BNPNPPPPNF 107:BNNNNPNPNF
108:BNPNNPPPNF 109:BPNNNPNNNF BPNPNPPPPF 111:BNNPPPPPPF
112: 112:BNPNNPNNNF 113:BNNPNPPPPF 114:BNPNNPPPPF 115:BPPNPPPPNF
116:BNPNPPNNPF 117:BNNNPPPNPF 118:BPNNPPPPPF 119:BPPNPPPPNF
120:BNNNNPPPNF 121:BNPNPNPPPF 122:BNPPPPNNNF 123:BNNNPPPNPF
124:BNPPNPNNNF 125:BNNNPPPNPF 126:BNNPPPPPPF 127:
F

  0:
```

Das Assemblerprogramm benötigt zu seiner Ausführung selbst einen Computer, der mit Lochstreifeneinheiten und einem Druckwerk ausgestattet ist. Im einfachsten Fall ist dies der eben zu programmierende Mikrocomputer, der mit einer Fernschreibmaschine verbunden ist. Kann ein anderer Computer verwendet werden, so hat dies fast immer den Vorteil der schnelleren Peripheriegeräte. Man spricht im zweiten Fall von einem Cross-Assembler. Mehrere Hersteller von Mikrocomputern bieten ihre Cross-Assembler auch zur Benützung über Time-Sharing-Terminals an.

8. Das Austesten

8.1 Simulation

Für einzelne Mikrocomputer werden sogenannte Simulatorprogramme angeboten, die ebenfalls auf dem Mikrocomputer selbst oder auf einem grösseren Computer betrieben werden können. Sie erlauben es, den Ablauf eines Programmes zu simulieren und an jeder beliebigen Stelle zu unterbrechen und dabei den Inhalt von Registern, den Programmzähler usw. ausschreiben zu lassen und allenfalls zu verändern. Simulatoren sind ein wertvolles Mittel, um ein Programm rasch auszutesten. Es ist jedoch zu beachten, dass der Test viel langsamer abläuft als das Programm im echten Betrieb. Das Verhalten des Programmes, zusammen mit den umgebenden Schaltungen, insbesondere in bezug auf das zeitliche Verhalten, kann nur im endgültigen System geprüft werden.

8.2 Austesten im System

Dazu muss das binäre Maschinenprogramm, wie es in Tabelle 6 gezeigt ist, erst einmal in den Programmspeicher gebracht werden. In einem Versuchsstadium kann der Programmspeicher als Schreib-Lese-Speicher ausgeführt sein. Über angeschlossene Geräte, zum Beispiel die bereits für das Assemblerprogramm benötigte Fernschreibmaschine, kann das Programm in den Speicher geladen und dessen Ausführung veranlasst werden.

Da Mikrocomputer keine Schalter und Anzeigen wie Minicomputer haben, ist es nur mit zusätzlichen Testhilfegeräten möglich, den Ablauf des Programmes auch zu überwachen. Vielfach ist der für das Programmieren eingesetzte Mikrocomputer in einem Gerät eingebaut, der diese Testhilfen bereits enthält (z.B. Intellec-Rechner von Intel). Für den späteren Test im Betrieb gibt es beispielsweise für den Rechner MCS-4 ein Gerät, das auf die integrierte Rechnerschaltung geklemmt werden kann, und auf dem recht viele Abläufe im Rechner auf Anzeigen überwacht werden können.

Die Programme müssen letztlich in Nur-Lesespeicher (ROMs) gebracht werden. Diese können teilweise mit Zusatzgeräten mit Hilfe des Mikrocomputers elektrisch programmiert werden. Im Prototypstadium werden dabei solche ROMs gebraucht, die mit ultraviolettem Licht wieder gelöscht werden können (RePROMs). Für grosse Serien werden ROMs verwendet, in die die Information bei der Herstellung mit Masken programmiert wird. Dies ist jedoch erst nach längerem praktischen Einsatz der Prototypen empfehlenswert.

9. Schlussbemerkungen

Nachdem Sie nun in der Lage sind, Mikrocomputer zu programmieren, sollten Sie auch einige Begriffe aus diesem Gebiet kennen, damit Sie sich mit Experten unterhalten können.

Mikroprozessoren sind diejenigen integrierten Schaltungen, die den eigentlichen Rechner enthalten, der im Mikrocomputer verwendet wird. Im Falle des Rechners MCS-4 besteht der Mikroprozessor aus einer einzigen integrierten Schaltung, dem Chip 4004.

Ist ein Mikroprozessor mit Taktgebern, Speicher und Interface-Schaltungen zu einer funktions- und kommunikationsfähigen Recheneinheit ergänzt, so spricht man von einem Mikrocomputer. Für den praktischen Einsatz dieses Mikrocomputers fehlen noch die Speisung und die anzuschliessenden Peripherieeinheiten.

Es ist jedoch falsch, vor alle Begriffe, die mit Mikrocomputern zu tun haben, ein «Mikro» zu setzen. So spricht man bei der Programmierung von Mikrocomputern nicht von Mikroprogrammierung. Unter diesem Begriff versteht man die Programmierung des Leitwerks eines Computers jeder Grösse. Bei der Mikroprogrammierung geht es darum, die sogenannten Makrobefehle, wie sie in obigen Beispielen verwendet wurden, in einzelne Mikrobefehle des Leitwerks an die Recheneinheit zu zerlegen. Von einem mikroprogrammierbaren Rechner spricht man aber nur dann, wenn die Möglichkeit besteht, durch Kombination des Leit- und Rechenwerks mit einem anderen Mikroprogramm, das in einem ROM gespeichert ist, den Rechner so zu verändern, dass er ein ganz anderes Repertoire von Makrobefehlen aufweist. Die wenigsten Mikrocomputer sind mikroprogrammierbar (z.B. National GPC-P und Intel 3000). Die wenigsten Anwender dürften mit der Mikroprogrammierung in Kontakt kommen.

Und nun frisch gewagt: Der Entwurf von Steuerungen unter Verwendung von Mikrocomputern oder gar Mikroprozessoren ist mindestens ebenso faszinierend wie mit herkömmlichen Elementen. Versuchen Sie es einmal auf dem Papier, wenn Sie die nächste Steuerung entwerfen, dann werden Sie die Vorteile der neuen Technik rasch erkennen. Auch wenn Sie merken sollten, dass ein Mikrorechner für ein bestimmtes Projekt nichts taugt, haben Sie gewonnen, nämlich mindestens die Erkenntnis, dass Sie von dieser Seite keine Konkurrenz befürchten müssen.

Die Qual der Wahl beim Mikroprozessor

oder Evaluationskriterien und Vergleich der Charakteristiken von erhältlichen und angekündigten Mikroprozessoren

Heute drei bis vier Jahre nach dem Erscheinen der ersten Mikroprozessoren – sind bereits mehrere dieser neuen Elemente verfügbar, eine grosse Anzahl ist angekündigt und soll in diesem oder nächsten Jahr ebenfalls erhältlich sein.
Die angebotenen Mikroprozessoren, mit Operandenwortlängen zwischen 4 und 16 bit, einem Adressierbereich von 4K bis 65K Worten und einer Instruktionsausführungszeit von 2 μs bis 44 μs, weisen sehr verschiedene Charakteristiken auf.
Bei der Evaluation von Mikroprozessoren sind die vorhandenen Architekturen wie Harvard-/Von-Neumann-Typ, Ein-/Multichip-CPU usw. sowie Parameter wie Instruktionsset, Mikroprogrammierbarkeit, Wortlänge, Adressierbereich, Register, Stack, Interrupt, synchron/asynchron usw. zu berücksichtigen. Auch Zweitlieferanten und vorhandene Softwareentwicklungshilfen sind von Bedeutung.
Beim Vergleich der verschiedenen Mikroprozessor-Parameter wirkt der uneinheitliche Sprachgebrauch in den Unterlagen verschiedener Hersteller erschwerend.

Aufgrund der Daten der lieferbaren und angekündigten Mikroprozessoren ist der folgende Trend feststellbar:

- *8 bit Operandenwortlänge*
- *65K Bytes-Adressbereich*
- *1-, 2-, (3)-Byte Instruktionswortlänge*
- *separater Adress- und Datenbus*
- *N-MOS-Technologie*
- *mehrere Adressiermodes zur Adressierung von Operanden und für Sprunginstruktionen*
- *mehrere vektorielle Interrupts*
- *DMA-Möglichkeit*
- *Standard RAM-, PROM- und ROM-Elemente für Speicher*
- *Bausteinsatz für I/O-Interfaces, Taktgenerator usw.*
- *Steigende Softwareunterstützung (Makroassembler, Debugprogramme, Software-User Club, PL/M-ähnliche High-Level Language)*

1. Einführung
1.1 Allgemeines

«Geschichte»:

Die ersten Mikroprozessoren (MCS 4004)[1] waren von Tischrechnern (Calculators) abgeleitet oder bekamen ihre Architektur von einem Hersteller intelligenter Terminals (MCS 8008).

Die ersten Mikroprozessoren wurden unter dem Schlagwort «Computer on a chip» angeboten, wobei ausser dem CPU-Chip nur einige Datenblätter zur Verfügung standen und der Anwender im übrigen auf sich selbst angewiesen war.

Das grosse Interesse, welches diese Elemente hervorrief, bewog verschiedene Halbleiterhersteller, Mikroprozessoren mit universelleren Strukturen zu entwerfen.

Ferner mussten Bereiche wie Ausbildung, Softwarehilfen, Mikrocomputersysteme (Hardwaresimulator) verbessert werden.

Während bei den ersten Mikroprozessoren der Akzent auf die Zentraleinheit gesetzt wurde, werden jetzt eine ganze Reihe von Zusatzbausteinen angeboten, welche Speicher, Input/Output-, Interruptsteuerung und sogar Interfaces an verschiedene periphere Geräte enthalten (Beispiel: Universal asynchronous Receiver/Transmitter, Interfaces an Display, Tastatur, Drucker, Diskette usw.). Einige Hersteller bieten bereits ganze Mikrocomputersysteme unter Anwendung von Mikroprozessoren an.

Die Hersteller konventioneller Minicomputer reagierten durch Anbieten von Single Board Processors, welche auf einer Platte Zentraleinheit und Minimalspeicher enthalten (Beispiel: α-16 von Computer Automation, LSI 11 von Digital Equipment). Langfristig wird der Unterschied zwischen Mikro- und Minicomputern verschwinden.

Es dürfte nicht übertrieben sein, wenn der technologische Wandel, welcher mit der Einführung der Mikroprozessoren verbunden ist, mit vergangenen Technologieänderungen wie Röhre/Transistor, Transistor/Integrierte Schaltung verglichen wird.

Personelle Probleme:

Das Arbeiten mit Mikroprozessoren erfordert gute Kenntnisse der Digitaltechnik (Hardware) und der Assemblerprogrammierung (Software). Vorteilhaft werden Hardware und Software zusammen entwickelt. In den meisten Entwicklungsabteilungen waren diese Bereiche bis anhin getrennt. In vielen Mikroprozessor-Anwendungen wird bisher festverdrahtete Logik ersetzt, deshalb sind die daran arbeitenden Entwicklungsingenieure meistens hardwareorientiert und müssen sich neu in die Grundlagen und Techniken der Software einarbeiten. Softwareleute, welche Erfahrungen mit grösseren Minicomputern besitzen, kommen nicht darum herum, sich um einzelne Bits in Interfaces und Prozessor zu kümmern und sich mit primitiveren Softwarehilfen, ohne Betriebssysteme, und langsameren peripheren Geräten zufriedenzugeben. Dabei stellen das einfachere Instruktionsset und die Verwendung von PROMs bzw. ROMs sowie Halbleiter-RAMs anstelle von Kernspeichern weitere Probleme.

Anwendungsbereich:

Das Hauptmotiv zur Anwendung von Mikroprozessoren liegt in der Möglichkeit, eine – im Vergleich zu festverdrahteter Logik – höhere Flexibilität zu erreichen.

Mit den gleichen Grundmoduln, jedoch mit verschiedenen Programmen (PROM), können unterschiedliche Varianten eines Systems relativ leicht implementiert werden.

Die Leistungsfähigkeit der Mikroprozessoren erlaubt in einigen Anwendungen auch den Ersatz konventioneller Minicomputer. Hier ist jedoch der auf wenige Tausend Franken zusammengeschrumpfte Preisunterschied der Hardware in Relation zum Softwareentwicklungsaufwand zu setzen. In den meisten Fällen dürfte – wenn die Gesamtkosten ermittelt werden – ein Einsatz von Mikroprozessoren erst ab einer minimalen Stückzahl annähernd gleicher Systeme sinnvoll sein.

1.2 Definitionen einiger Begriffe

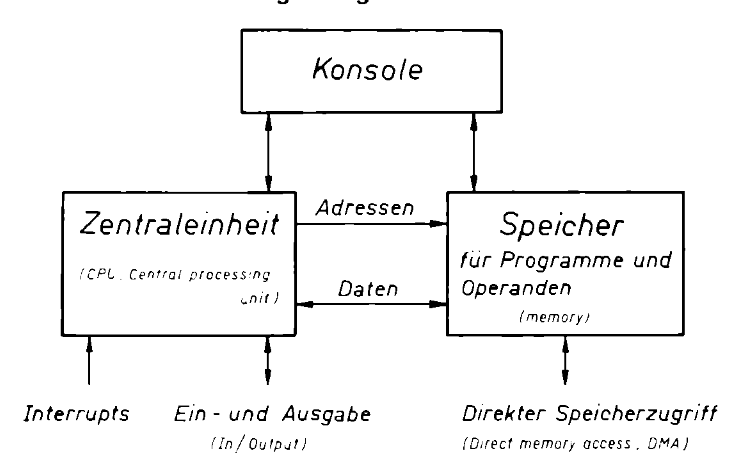


Bild 1 Grundkonzept eines Computers.
Ein Computer besteht aus einer Zentraleinheit, einem Speicher – welcher Programme (Arbeitsvorschriften) und Daten enthält – sowie Ein/Ausgabekanälen – welche die Verbindung mit der Aussenwelt herstellen. Die Konsole stellt das Bedienungsorgan des Computers dar; bei einigen Mikrocomputern wird die Konsole durch eine Ein/Ausgabeschreibmaschine (Teletype) realisiert.

Mikrocomputer oder Mikroprozessoren bestehen aus einem oder mehreren – durch Programme steuerbaren – LSI-Komponenten, welche folgende Funktionen erfüllen können:

- arithmetische und logische Operationen mit Daten
- datenabhängige Entscheidungen hinsichtlich des Programmablaufs
- Ein- und Ausgabe von Daten, Zustands- und Steuerinformationen

In der Literatur und den Angaben der Hersteller wird selten klar unterschieden, ob unter dem Begriff Mikrocomputer nur die Zentraleinheit (CPU) oder das gesamte System, d.h. Zentraleinheit inkl. Speicher, Konsole, In/Outputadapter usw. enthalten ist.

Im folgenden wird für die integrierte Zentraleinheit der Begriff Mikroprozessor und für das gesamte System die Bezeichnung Mikrocomputer gewählt.

Es werden folgende *Speicherarten*[2] unterschieden:

RAM, Random Access Memory:
Schreib-/Lesespeicher mit Zugriff zu beliebigen Speicherzellen

ROM, Read Only Memory:
Lesespeicher. Der Speicherinhalt wird bei der Herstellung des Speichers via Masken festgelegt.

PROM, Programmable Read Only Memory:
Lesespeicher der vom Anwender programmiert werden kann, dabei werden die einzelnen bits des Speichers mittels «Durchbrennen» von Widerständen oder Diodenstrecken oder durch Aufladen von Kondensatoren mit extrem hoher Entladezeit programmiert.

RePROM, Reprogrammable Read Only Memory:
Reprogrammierbarer, bzw. löschbarer Lesespeicher. Das Löschen

[1] siehe z. B. K. Wüthrich: «Mikrocomputer» und T. Kaegi: «Das Programmieren von Mikrocomputern», Seiten 9ff bzw. 39ff dieser Broschüre.

[2] siehe z. B. auch R. Zinniker: «Digitale Halbleiterspeicher», Seiten 23ff dieser Broschüre.

des gesamten Speichers (einzelne Zellen sind nicht löschbar) erfolgt durch Bestrahlung mit UV-Licht. Damit werden die bei der Programmierung aufgeladenen Kondensatoren entladen. Es ist ein mehrfaches Programmieren und Löschen möglich.
PLA, Programmable Logic Array:
PLAs sind ROMs, deren Codierlogik zum Adressieren der Speicherzellen ebenfalls programmierbar ist. Beispielsweise die Auswahl von 2^n Zellen aus m Adressleitungen, wobei $n < m$.
«Non volatile» RAMs:
Schreib/Lesespeicher mit beliebigem Zugriff, welche ihre Information bei Netzspannungsausfall beibehalten. Halbleiterspeicher verlieren – im Gegensatz zu Kernspeichern – ihre Information bei einem Spannungsausfall. Es ist zu erwarten, dass in einigen Jahren auch non volatile Halbleiter-RAMs verfügbar sein werden. Bei echten «non volatilen» RAMs sollten die elektrischen Ein/Ausgangscharakteristiken beim Schreiben und Lesen nicht verschieden sein.
EAROMs (Electrically Alterable Read Only Memories) beziehungsweise RMM (Read Mostly Memories)
verhalten sich ähnlich wie die «non volatilen» RAMs, doch müssen beim Schreiben andere Spannungen, Ströme und zeitliche Bedingungen eingehalten werden als beim Lesen.
Bei Speichern wird ferner zwischen *«destructive»*- und *«non destructive»*-Readout unterschieden. Beim Kernspeicher geht die Information durch das Lesen verloren, d.h. nach dem Lesen muss unmittelbar ein Wiedereinschreiben erfolgen. Dieser Vorgang wird stets durch die Kernspeicherzusatzelektronik gesteuert.
Halbleiterspeicher weisen «non destructive»-Readout auf.
Bei der Hilfssoftware (Hilfsprogramme) werden folgende Begriffe verwendet:
Resident Software/Cross Software:
Residente Programme laufen auf dem Mikrocomputer, während Cross-Programme auf einem anderen (meistens grösseren) Computer, beispielsweise auf einem PDP 11-Minicomputer, arbeiten.
Assembler:
Programm zum Übersetzen der Mnemocodes (Instruktionen), symbolischen Adressen und Operanden in den Maschinencode. Prüfen das Quellenprogramm (Source) auf syntaktische Richtigkeit.
Der Cross Assembler kann auf einem Minicomputer laufen, er erzeugt einen Lochstreifen («Maschinensprache»), der mit einem Ladeprogramm auf dem Mikrocomputer geladen werden kann.
Compiler:
Programm zum Übersetzen einer höheren Programmiersprache (z.B. PL/M) in Maschinensprache. Läuft meistens auf einem grösseren Computer.
Editor:
Programm, welches das Eintippen von Quellenprogrammen unterstützt (Erstellen eines maschinell lesbaren Datenträgers).
Es erlaubt Textmanipulationen zum effizienten Korrigieren, Verbessern und Erweitern von Quellenprogrammen.
Simulator:
Erlaubt das Simulieren eines Mikroprozessors auf einem grösseren Computer. Unterstützung des Austestens von Programmen. Die Ein-/Ausgabe muss ebenfalls simuliert werden.
Debugging (Entlausen, Ausprüfen/-testen):
Unterstützt das Austesten von Programmen auf dem Mikrocomputer. Einsetzen von Breakpoints (Anhaltepunkte), Betrachten und Ändern des Inhalts von Speicherzellen und Registern usw.
Lader:
Dient zum Laden von übersetzten Programmen von einem maschinell lesbaren Datenträger (z.B. Lochstreifen) in den Arbeitsspeicher des (Mikro-)Computers.

1.3 Entwicklung der Integration

Die heutigen LSI-Bausteine (Mikrocomputer, Speicher) enthalten in der Grössenordnung 1000 bis 10000 Schalterfunktionen (Transistoren).
Vergleichsweise enthielt einer der ersten elektronischen Computer, der zwischen 1943 und 1946 von John Mauchly und Prosper Eckert gebaute ENIAC, 18000 Elektronenröhren und ca. ½ Million handgelöteter Lötstellen. Er wog 30 Tonnen, benötigte eine Fläche von 150 m² und verbrauchte eine Leistung von 174 kW. Die Programmierung erfolgte über 6000 Schalter und ein Steckbrett.
Daraus wird ersichtlich, dass für die in einem Mikrocomputersystem implementierten Funktionen vor ca. 30 Jahren noch ein grösserer Raum mit vielen Schränken und Ventilatoren erforderlich war.
Was heute bequem per Post in kleinen Schachteln geliefert wird oder in der Hosentasche mitgenommen werden kann, hätte früher bei gleichwertiger Funktionskomplexität einen Lastwagenzug benötigt.

1.4 Versuch einer Klassifizierung von Mikro- und Minicomputern

Obwohl in vielen Fällen die Übergänge zwischen den einzelnen Gruppen fliessend sind, bilden die im folgenden genannten Gruppen Schwerpunkte in dem gesamten Spektrum der Mikro- und Minicomputer.

Tabelle 1 Klassifizierung von Mikro- und Minicomputern

Bezeichnung	Eigenschaften	Beispiele
Taschenrechnerchips	Vorwiegend für BCD-Arithmetik ($n \cdot$ 4-bit-Operanden) geeignet, wenig logische Operationen. Fest implementierte (Firmware) Grundoperationen für +, −, ×, :, Programmablauf durch Tastenbetätigung gegeben	Texas TMS 0100
4 bit Mikrocomputer	Abgeleitet von Taschen- und Tischrechnerchips mit Erweiterungen für logische Operationen, oft relativ unübersichtliche Architektur, komplizierte Speicheradressierung	MCS 4004 MCS 4040 PPS4 IMP4
8 bit und 16 bit-Mikrocomputer	Single und Multichipsets, 2- bis 10mal langsamer als Minicomputer, begrenzte Leistung des Instruktionssets im Vergleich zu Minicomputer, begrenzte Soft- und Hardwareunterstützung in Vergleich zu Minicomputer	MCS 8008 MCS 8080 MC 6800 PIP, PPS8 IMP16 usw.
Singleboard Minicomputer (LSI-Minicomputer)	Softwarekompatibel mit vorhandenem Minicomputer	α 16 LSI 11 usw.
Minicomputer	Gesamte Hardware- (Peripherie, Prozessinterface) und Softwareunterstützung (Betriebssystem, Übersetzer, Editor usw.) vorhanden	PDP11 NOVA HP 2100 usw.

2. Grundorganisationen von Computern

Bild 2 zeigt die wichtigsten Grundorganisationen, wobei Bild 2a allgemein mit «Harvard»- und Bild 2b mit «Von-Neumann»-Computer bezeichnet wird:

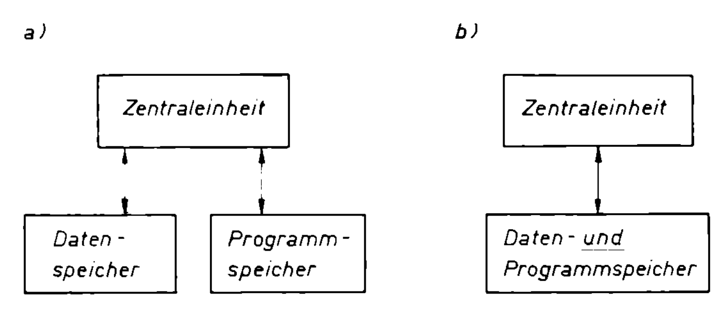


Bild 2 Grundorganisationen von Computern.
Beim «Harvard» Konzept (Bild 2a) sind Programm- und Datenspeicher getrennt.
Die heute meistens verwendete «Von Neumann»-Organisation (Bild 2b) verwendet einen gemeinsamen Programm- und Datenspeicher.
Trotzdem ist bei einigen Mikrocomputern die Harvard-Organisation zugrundegelegt (z.B. MCS 4004).

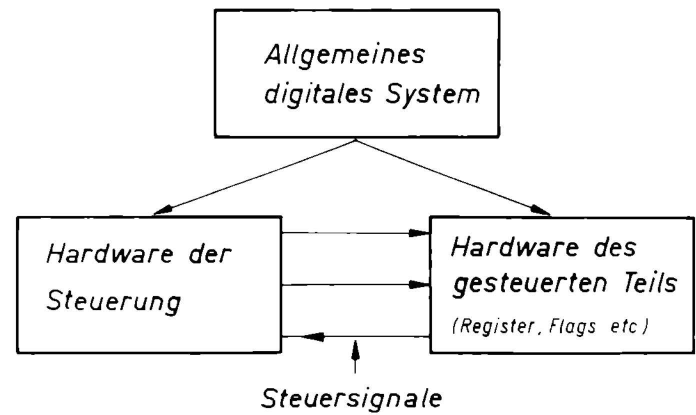


Bild 3 Jedes digitale System – bestehend aus kombinatorischer und sequentieller Logik – kann funktionell in einen Steuer- und einen gesteuerten Teil aufgetrennt werden. Im Gegensatz zur festverdrahteten Steuerung, die aus einer massgeschneiderten Kombination von logischen Toren und Flip-Flops besteht, wird bei einer mikroprogrammierten Steuerung die in Bild 4 und Bild 5 dargestellte, allgemein gültige Struktur verwendet. Dabei werden die auszuführenden Funktionen in mehrere Elementarschritte aufgeteilt (Mikroinstruktionen), welche nach einer programmierten Sequenz (Mikroprogramm) ablaufen.

Das Konzept gemäss Bild 2b wurde etwa 1945 von «Von Neumann, Mauchly und Eckert» festgelegt und hat sich bei den Mini- und Grosscomputern vollständig durchgesetzt.
Hingegen wurde in neuester Zeit bei 4-bit-Mikroprozessoren (z.B. MCS 4004, MCS 4040 von INTEL) wiederum die Harvard-Architektur verwendet. Die Harvard-Architektur erlaubt verschiedene Wortlängen für Daten und Instruktionen, dies ist jedoch auch in der Von-Neumann-Architektur möglich, wenn Multiwort-Instruktionen verwendet werden (wobei unter einem Wort eine adressierbare Speicherzelle verstanden wird).
Z.B. 8 bit für Datenworte; 8, 16, 24 bit beziehungsweise 1, 2, 3 Worte für Instruktionen.
Da 4 bit zur Codierung von Instruktionen nicht genügen, werden bei 4-bit-Mikroprozessoren (4-bit-Wortlänge für Operanden) Instruktionen mit 8 bis 20 bit verwendet.
Die Von-Neumann-Architektur ermöglicht eine programmgesteuerte Änderung des Programms. Da Instruktionen und Daten in gleicher Weise modifiziert werden können, erlaubt dies sehr viele Möglichkeiten in der Programmierung und führt in der letzten Konsequenz zum «lernenden» Automaten.
(In der Praxis ist die Änderung von Instruktionen durch das Programm sehr verpönt.)
Bei Anwendungen von Mikroprozessoren ist diese Eigenschaft jedoch weniger von Bedeutung, da die Programme meist in ROMs abgespeichert werden und lediglich variable Daten in RAMs enthalten sind. Die Daten können natürlich den Programmablauf beeinflussen (bedingte Sprungfunktionen), sie sind jedoch nicht als eigentliche Instruktionen zu betrachten.

3. Mikroprogrammierung von Mikroprozessoren
3.1 Was ist Mikroprogrammierung?

Die Idee der Mikroprogrammierung wird einer Arbeit von M.V. Wilkes [1] im Jahre 1951 zugeschrieben; die Methode setzte sich jedoch erst 10 bis 20 Jahre später durch, als billige, vor allem Halbleiterspeicher zur Verfügung standen [2].
Es handelt sich um eine Konzeption für die Realisierung von komplexen Schaltungsaufgaben, welche vor allem beim Aufbau von Zentraleinheiten (CPU) angewendet wird. Während in den älteren – nicht mikroprogrammierten – Zentraleinheiten der Ablauf einer Maschineninstruktion durch eine (massgeschneiderte) Kombination von Toren, Flip-Flops usw. gesteuert wurde, wird dieser Ablauf bei den mikroprogrammierten Systemen durch eine Folge von Mikroinstruktionen (Mikroprogramm) gesteuert (Bilder 3, 4 und 5).
Die Ausführung einer Instruktion wird somit durch Auslösen eines Mikroprogramms gestartet, wobei die Startadresse durch den Operationscode der Instruktion gegeben ist. Unter Instruktion wird im folgenden stets eine Maschineninstruktion (Assemblerinstruktion) verstanden, diese Instruktionen befinden sich im Arbeitsspeicher des Computers. Das Mikroprogramm einer Instruktion besteht im wesentlichen aus einem Ausführungsteil (Execute-Cycle) und einem Teil zur Adressberechnung und Lesen der nächsten Instruktion (Fetch-Cycle).
Im Ausführungsteil werden vom Mikroprogramm zusammen mit der Mikroprogrammsteuerlogik (Microprogram control unit) der Transfer von Daten zwischen Registern oder Register und Arbeitsspeicher gesteuert, ferner werden für arithmetische und logische Instruktionen die notwendigen Signale für die ALU (Arithmetic Logic Unit, diese enthält Addierwerke, Shifter usw.) erzeugt. Nach der Ausführung wird die Adresse der nächsten Instruktion ermittelt, wobei bei Sprunginstruktionen (bedingte oder unbedingte) der Operandenteil der Instruktion und bei bedingten Sprunginstruktionen zusätzlich das Resultat des Ausführungsteils (z.B. bei JUMP IF ZERO) berücksichtigt wird.
Man unterscheidet zwischen monophaser und polyphaser Mikroprogrammierung, oft werden auch die Ausdrücke vertikale und horizontale Mikroprogrammierung verwendet [3].
Bei monophaser (vertikaler) Mikroprogrammierung läuft eine Mikroinstruktion in einem Taktzyklus ab; für die Steuerung einer Maschinenkonstruktion werden viele Mikroinstruktionen mit relativ geringer Wortlänge benötigt.
Bei polyphaser (horizontaler) Mikroprogrammierung werden lange Mikroinstruktionen und mehrere Taktzyklen zu deren Ausführung benötigt.
Die Anzahl der Mikroinstruktionen ist geringer, hingegen die Überschaubarkeit (Timingprobleme!) schwieriger als bei monophaser Mikroprogrammierung.

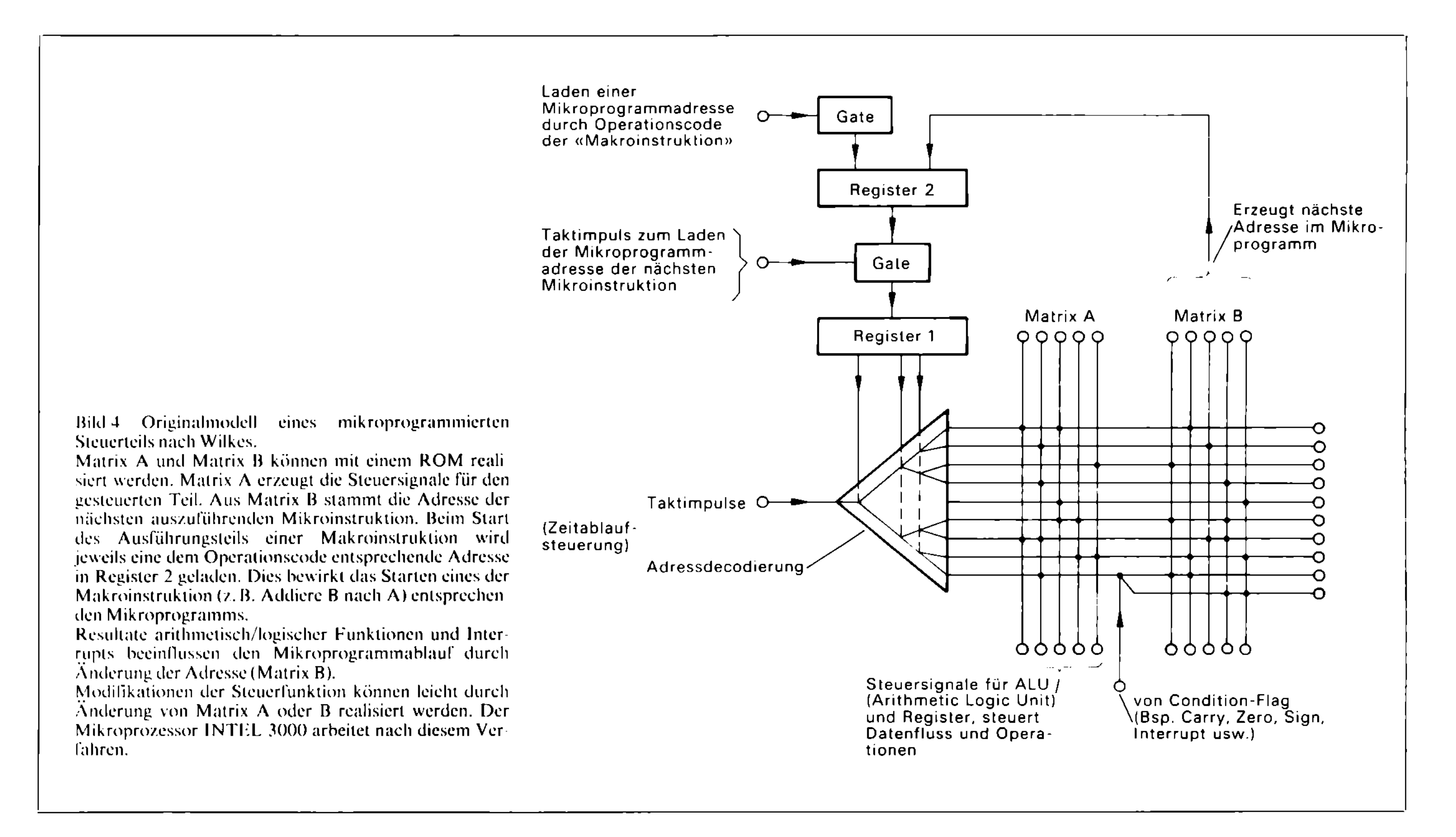


Bild 4 Originalmodell eines mikroprogrammierten Steuerteils nach Wilkes.
Matrix A und Matrix B können mit einem ROM realisiert werden. Matrix A erzeugt die Steuersignale für den gesteuerten Teil. Aus Matrix B stammt die Adresse der nächsten auszuführenden Mikroinstruktion. Beim Start des Ausführungsteils einer Makroinstruktion wird jeweils eine dem Operationscode entsprechende Adresse in Register 2 geladen. Dies bewirkt das Starten eines der Makroinstruktion (z. B. Addiere B nach A) entsprechenden Mikroprogramms.
Resultate arithmetisch/logischer Funktionen und Interrupts beeinflussen den Mikroprogrammablauf durch Änderung der Adresse (Matrix B).
Modifikationen der Steuerfunktion können leicht durch Änderung von Matrix A oder B realisiert werden. Der Mikroprozessor INTEL 3000 arbeitet nach diesem Verfahren.

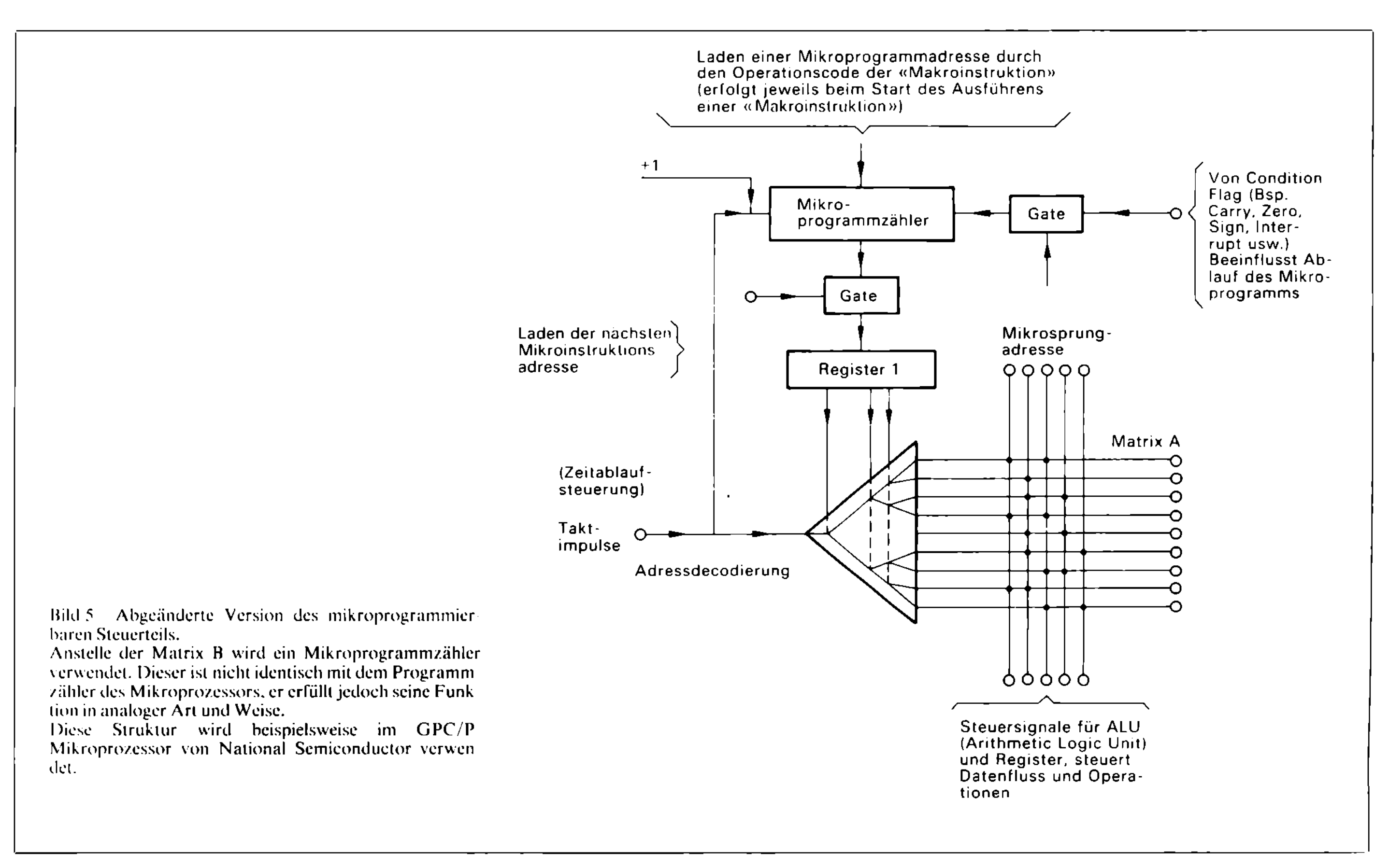


Bild 5 Abgeänderte Version des mikroprogrammierbaren Steuerteils.
Anstelle der Matrix B wird ein Mikroprogrammzähler verwendet. Dieser ist nicht identisch mit dem Programmzähler des Mikroprozessors, er erfüllt jedoch seine Funktion in analoger Art und Weise.
Diese Struktur wird beispielsweise im GPC/P Mikroprozessor von National Semiconductor verwendet.

Die Ausführungszeit einer Instruktion hängt von der Anzahl notwendiger Mikroinstruktionen (2 bis 3 Mikroinstruktionen für Register Transfer-Instruktionen, einige 100 Mikroinstruktionen für Multiply/Divide Floatingpointinstruktionen) und damit auch von der Zugriffszeit des Mikroprogrammspeichers ab. Deshalb ist eine möglichst kleine Zugriffszeit des Mikroprogrammspeichers erwünscht, z. B. einige 10 ns bei Grosscomputern und Minis, einige 100 ns bei MOS-Mikrocomputern.
Die Wortlänge der Mikroinstruktionen liegt bei Mikroprozessoren in der Grössenordnung 18 bis 24 bit, bei Grosscomputern werden Wortlängen von 60 bis einige 100 bit verwendet.
Beispielsweise ist das Instruktionsset des Mikroprozessors IMP16C von National Semiconductor in 100 Mikroinstruktionen zu 23 bit implementiert. Für Erweiterungen dieses Instruktionssets kann ein zweites oder drittes Mikroprogramm-ROM vorgesehen werden.
Bild 4 zeigt das von Wilkes vorgeschlagene Modell. Das Mikroprogramm ist in Matrix A und in Matrix B enthalten. Während Matrix A die Steuersignale für die ALU generiert, erzeugt Matrix B die Adresse der nächsten Instruktionen. Von der ALU oder von aussen (z. B. Interrupt) werden Sprungbedingungen in die Mikroprogramm-Steuereinheit zurückgeleitet. Bild 5 zeigt eine abgeänderte Version, welche anstelle der Matrix B einen Mikroprogrammzähler enthält, der entweder inkrementiert oder vom Mikroprogramm (Mikroprogramm-Branch) oder von der Instruktion (Operationscode bildet Startadresse des Mikroprogramms) geladen werden kann.
Das zwischen Hard- und Software liegende Resultat der Mikroprogrammierung wird oft mit *Firmware* bezeichnet.

Tabelle 2 Effizienzvergleiche von Mikro- und (Makro-)Programmierung am Beispiel des Mikroprozessors IMP16 von National Semiconductor [4]. Die Programmeffizienz kann durch die benötigte Speichergrösse (Anzahl bit) und die Ausführungszeit charakterisiert werden.

Aufgabe	(Makro)-programmierung		Mikro-programmierung	
	Anzahl Instr. zu 16 bit	Ausführungszeit in us[1]	Anzahl u-Instr. zu 23 bit	Ausführungszeit in us[2]
Interruptscanning: Bei einer Sammelinterruptleitung an der bis zu 16 Interrupt-Flags angehängt werden können, sind die Interruptverursacher zu lokalisieren (Abfrage, Lokalisierung der Flags, Bildung einer entsprechenden Sprungadresse [Trap Vektor]). Bei der mikroprogrammierten Version wird für die Abfrage Shift-, Testoperationen usw. lediglich 1 (Makro-)Instruktion benötigt.	32	530	11	112
Multiplikationen	11	678	?	171

[1] ohne Berücksichtigung der zusätzlichen Erhöhung bei Verwendung von Arbeitsspeichern mit grosser Zugriffszeit (z. B. 1 us)
[2] 1 Microcycle = 1,4 us

3.2 Vorteile der Mikroprogrammierung für den Computerhersteller

1. Reduktion der kombinatorischen und sequentiellen Logik und damit übersichtlicheres, strukturiertes Konzept.
2. Erweiterungen des Instruktionssets können relativ leicht implementiert werden, dabei ist im Vergleich zur Implementierung der gewünschten Funktionen in Form von Subroutinen im Makroinstruktionsset eine Einsparung an Speicherbedarf und Ausführungszeit möglich (siehe Tabelle 2). Bei vielen Computern wurden nachträglich Multiplikation, Division, Floatingpoint-Arithmetik, höhere Ein/Ausgabeinstruktionen durch Erweiterung des Mikroprogramms realisiert.
3. Emulation des Instruktionssets von anderen Computerherstellern oder früheren Modellen des gleichen Herstellers.

Grosscomputer und einige Minicomputer enthalten teilweise einen schreibbaren Mikroprogrammspeicher (WCS, Writable Control Storage). Damit wird es möglich, das Instruktionsset des Computers durch Laden eines neuen Mikroprogramms (ab Kassette oder Diskette) in den Mikroprogrammspeicher zu ändern.
Beispielsweise werden mit dieser Methode Programme der alten IBM-Modelle 1401 oder 7090 auf einer IBM-360-Maschine zum Laufen gebracht, ebenfalls können Programme von IBM-Maschinen auf entsprechenden WCS-Maschinen von UNIVAC arbeiten.

3.3 Mikroprogrammierung durch den Anwender von Mikroprozessoren

Obwohl die Zentraleinheiten von vielen Mikroprozessoren mikroprogrammiert sind, werden zurzeit nur das Mikroprozessorsystem GPC/P von National Semiconductor sowie das INTEL-3000-System vom Hersteller für die Mikroprogrammierung durch den Anwender unterstützt. (Die Mikroprogrammierung von Minicomputern wird vor allem von Hewlett Packard unterstützt.)

Vorteile für den Anwender:
Das Instruktionsset kann den Anforderungen des Anwenders angepasst werden, damit ist eine Verbesserung hinsichtlich Zeit- und Speicherbedarf erreichbar.
Beispiele von Aufgaben (Instruktionen), welche im Mikroprogramm implementiert werden können:
- Monitorfunktionen
- On line Diagnostikfunktionen
- Veränderung der Operandenlänge (z. B. multiple precision)
- Veränderung der Interruptstruktur (z. B. vektorielle Interrupts mit vielen Prioritätsebenen, Softwareinterrupt)

Es kann ein – zu einer bisher verwendeten Maschine – aufwärtskompatibles Instruktionsset entwickelt werden, falls eine gewisse Ähnlichkeit der Architektur der beiden Maschinen vorhanden ist. Damit wird es möglich, die bereits entwickelte Software zu übernehmen.

Nachteile:
Aufwand: die Mikroprogrammierung setzt grosse Detailkenntnisse des Mikroprozessorsystems sowie eine gute Übersicht der Einflüsse des Instruktionssets auf die Lösung der Anwenderprobleme voraus. Einarbeitungs-, Entwicklungs- und Debugzeit sind hoch. Da auch die Maskenkosten des kundenspezifischen Mikroprogrammspeichers (ROM) hoch sind, dürfte sich eine Mikroprogrammierung durch den Anwender bei einem Bedarf unter 1000 Computern kaum lohnen.
Die vom Hersteller des Mikroprozessors erhältliche Software (Assembler, Editor, Compiler, Debugprogramme, Hilfsroutinen [z. B. mathematisches Programmpaket]) sind nicht oder nur teilweise verwendbar.

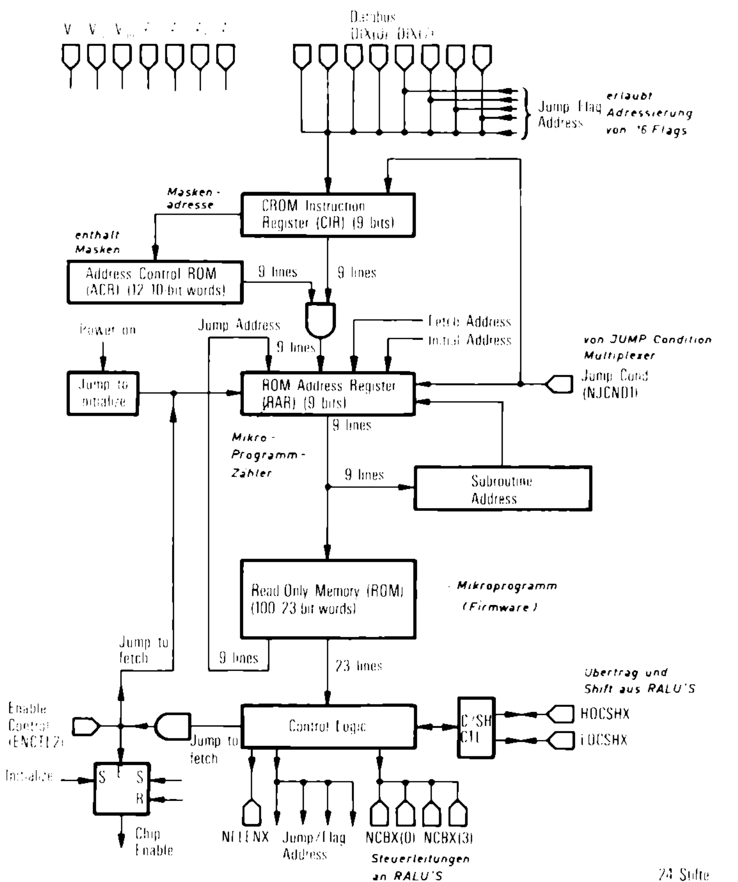


Bild 6 Control Read Only Memory (CROM) des GPC/P Mikroprozessors von National Semiconductor (24-Stift-Gehäuse). Enthält einen Mikroprogrammzähler (RAR), der inkrementiert oder vom Mikroprogramm-Subroutinen-Adressregister (SAR) oder durch den Operationscode der Makroinstruktion geladen werden kann.
Die Übertragung der Steuersignale an die Registereinheit (RALU) erfolgt in vier Taktzeiten Φ1, Φ2, Φ3, Φ4.
Der Operationscode gelangt nicht direkt in den Mikroprogrammzähler (RAR): über das Adress Control ROM erfolgt eine Maskierung (AND Funktion). Das Subroutinenadressregister (SAR) dient zum Zwischenspeichern der Rücksprungadresse bei Aufruf von Mikroprogrammsubroutinen.

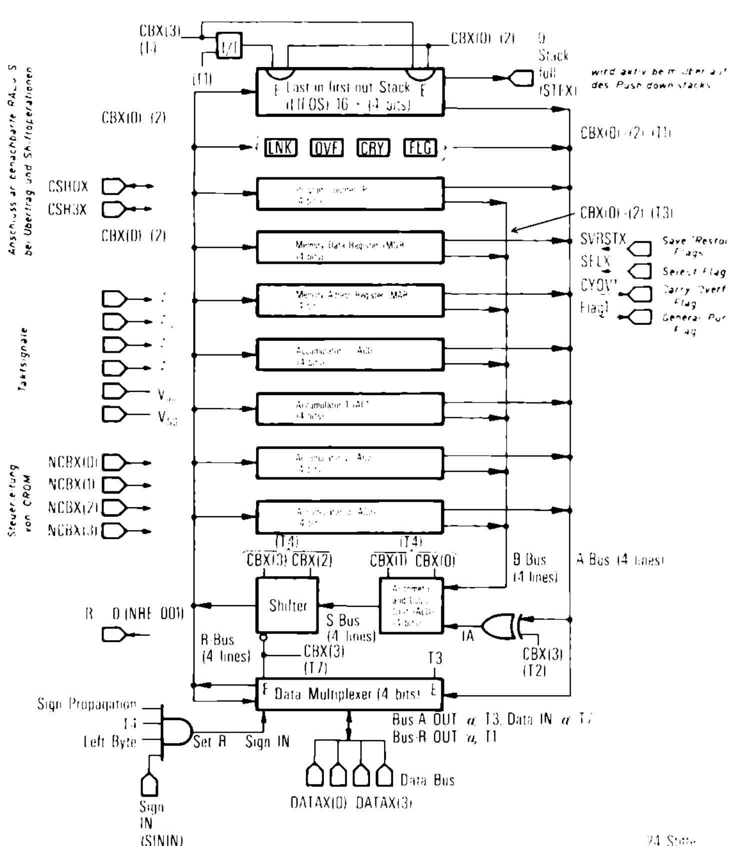


Bild 7 Registereinheit, Addier- und Shift-Elemente (RALU) des GPC/P Mikroprozessors von National Semiconductor (24-Stift-Gehäuse).
Bei dieser RALU handelt es sich um eine 4-bit-Einheit (4 bit slice), d.h. alle Register weisen eine Wortlänge von 4 bit auf.
In der dargestellten Anordnung werden 4 Register als Akkumulatoren verwendet, ferner ist je ein Register zur Speicherung von Operandenadresse (MAR), Instruktionsadresse (PC), Daten (MDR) und Control Flags (LNK, OVF, CRY, FLG) vorhanden. Der interne Stack hat eine Kapazität von 16 × 4 bit.
Die Steuerung der RALU erfolgt via die Signalleitungen NCBX(...3) in Vier-bit-Paketen vom CROM während den Taktsignalen Φ1, Φ2, Φ3, Φ4.
Der bidirektionale Datenbus DATAX (0...3) dient zum Übertragen von Daten und Adressen zwischen RALUs und Arbeitsspeicher, beziehungsweise Peripherie.

3.4 Beispiel eines mikroprogrammierbaren Mikroprozessors (GPC/P von National Semiconductor)

Die Bilder 6 und 7 zeigen den Aufbau der Mikroprogrammsteuereinheit (CROM), beziehungsweise der Registereinheit (RALU), des GPC/P-Systems.
8 Bits des Operationscodes der Instruktion und ein 9. Bit, das von der Registereinheit (z.B. Zero-, Carry Flag usw.) oder von aussen (Interrupt) stammen kann, werden via DIX(0) bis DIX(7), NJCND1 in das CROM-Instruktionsregister geladen und ergeben nach einer Maskierung durch das Address-Control-ROM (ACR) die Startadresse des einer Instruktion zugeordneten Mikroprogramms. Die Adressinformationen der Instruktion (Register- oder Speicheradressierung) gelangen direkt in die Registereinheit (RALU).
Das Mikroprogramm ist in einem ROM von 100 × 23 bit enthalten. Für notwendige Erweiterungen des Instruktionssets können entsprechend dem Adressierbereich von 9 bit bis zu 5 CROMs «parallel» geschaltet werden.
Ein Subroutinenadressregister ermöglicht die Bildung von Mikroprogrammsubroutinen (1 Nesting level).
CROMs und RALUs arbeiten mit einem 4-Phasen-Takt (Φ1, Φ3, Φ5, Φ7). Die Steuersignale zwischen CROM und RALU werden im Zeitmultiplexbetrieb (4 Takte während eines Mikrozyklus) in 4 × 4-bit-Paketen via die Anschlüsse NCBX(0) bis NCPX(3) übertragen.

Für die Entwicklung von Mikroprogrammen durch den Anwender ist von National Semiconductor ein auf einer Printplatte aufgebauter Modul mit der Bezeichnung FACE (Field alterable control element) erhältlich. Dieser Modul ist funktionell identisch zum CROM, erlaubt jedoch für den Mikroprogrammspeicher die Anwendung von Standard-RAM- oder -PROM-Elementen. Die Mikroprogrammentwicklung wird ausserdem von einem Mikroprogramm-Assembler unterstützt.

Literaturverzeichnis

[1] The best way to design an automatic calculating machine, Manchester University Computer Inaugural Concerence 1951, S.16. – [2] Design of microprogrammable Systems, W.H. Davidow, Signetic Application Notes July 73, SMS 0052 AN. – [3] Microprogramming: More 'in' than ever, D.R. Lewis, Electronic Design 17, Aug. 16, 1973, S. 58–63. – [4] Extend LSI processor capabilities, G. Reyling, Electronic Design 22, Oct. 25, 1974, S. 90–95. – [5] Mikroprogrammierung, P. Schnupp, H.L. Wieler, Bürotechnik 3/72, S. 380–386. – [6] The Role of Technology in Microcomputer Design and Evolution, F. Faggin, Circuit and Systems, Febr. 75, S. 4–13. – [7] In Switch to N MOS microprocessor gets a 2 µs cycle time, M. Shima, F. Faggin, Electronics, April 18, 74, S. 95–100. – [8] Microprocessor Score card, J. Odgin, Euromicro News letter, Jan. 75, Nr. 2, S. 43–77. – [9] Microprocessor and microcomputer survey, D.J. Theiss, Datamation, Dez. 74, S. 90–101. – [10] Focus on microprocessors, E.A. Torrero, Electronic Design 18, Sept. 1, 74, S. 52–69. – [11] Current Microcomputer Architecture, R.M. Holt, M.R. Lemas, Computer Design, Febr. 74, S. 65–73. – [12] Introduction aux microprocesseurs, J.D. Nicoud, Paper von Journées d'électronique EPFL, Lausanne 1974, S. 1–13. – [13] Utilities for the development of Software for microcomputers, P. Schneider, Paper von Journées d'électronique EPFL, Lausanne 1974, S. 185–194. [14] High Level Language simplifies microcomputer programing, G.A. Kildall, Electronics, June 27, 1974, S. 103–109. – [15] Cross-Assembler für den Mikrocomputer MCS 8080 auf einer PDP11-Anlage, Paper von MIMIC 75, S. 121–123.

4. Einchip-, Multichip-Mikroprozessoren

4.1 Allgemeines

Zurzeit werden Mikroprozessoren angeboten, bei denen die wichtigsten Funktionen der Zentraleinheit mit einem LSI-Element (Computer on a chip, z. B. INTEL MCS 8008, INTEL MCS 8080, Motorola MC 6800, Rockwell PPS 8 usw.) oder mit einigen (3 bis 6, einige bis 10) LSI-Elementen (Multichip-Mikroprozessoren, z. B. GPC/P, bzw. IMP 4, IMP 8, IMP 16 von National Semiconductor, INTEL-3000-System usw.) realisiert werden.

Natürlich werden – zur Erfüllung der gesamten Funktion der Zentraleinheit – zusätzlich zu diesen LSI-Elementen, eine von Hersteller zu Hersteller variierende Zahl von SSI-Bausteinen für Taktgeneratoren, Decoder, Adressregister, Sprungkonditionsmultiplexer, Interruptprioritätsschaltungen, Buffer usw. benötigt.

4.2 Einchip-Mikroprozessor (Computer on a chip)

Die meisten Einchip-Mikroprozessoren der neueren Generation haben ein 40-Stift-Gehäuse und weisen – bei einigen herstellerbezogenen Abweichungen – die in Bild 8 dargestellte Struktur auf.

4.3 Multichip-Mikroprozessoren

Bild 9 zeigt die innere Organisation von Multichip-Mikroprozessoren.

Die *Registereinheit* enthält Akkumulatoren, Control-Flags (Carry-, Zero-, Sign-Flag usw.), Indexregister, Programmzähler, Adressregister, Stackpointer oder interner Push down Stack sowie die arithmetisch-logische Steuerschaltung (Arithmetic Logic Unit, ALU, diese enthält Addierwerke, Shifter usw.)

Bei einigen Herstellern wird die Registereinheit in 2- oder 4-bit-«Slices» aufgeteilt. Damit können Mikroprozessoren mit verschiedener Wortlänge aufgebaut werden (z. B. GPC/P-System von National Semiconductor erlaubt Wortlängen von 4, 8 bis 32 bit, INTEL 3000 bis 320 bit).

Die untere Grenze der Wortlänge ist durch den Adressierbereich gegeben, deshalb war es beim 4-bit-(IMP 4) und 8-bit-Rechner (IMP 8) von National Semiconductor notwendig, die Länge des Programmzählers und Stacks durch externe Elemente zu erweitern.

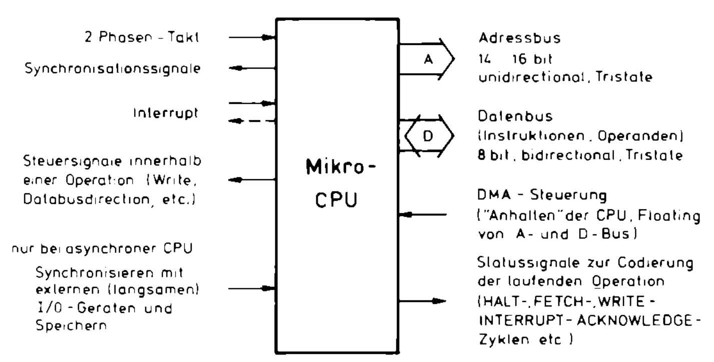


Bild 8 Einchip Prozessor
Die Signale zur Aussenwelt dienen zum Adressieren von Instruktionen und Operanden (A Bus), zum Transfer dieser Daten (D Bus) und zur Steuerung des Zeitablaufs sowie zum Erkennen von Interrupts.

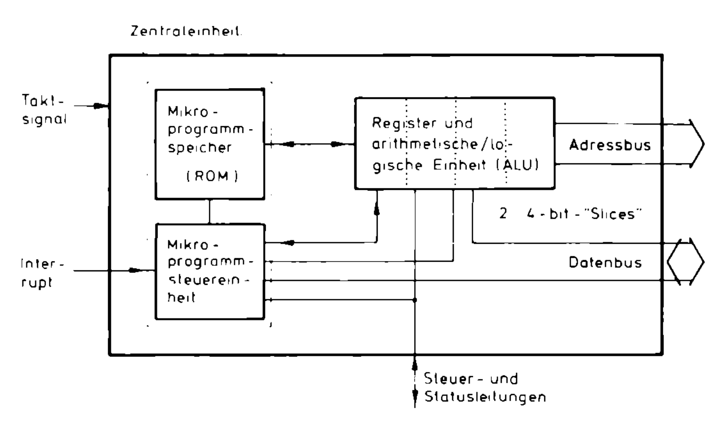


Bild 9 Ein Multichip-Prozessor kann in die Funktionselemente:
Register und arithmetisch/logische Einheit (ALU)
Mikroprogramm Steuereinheit
Mikroprogramm Speicher
aufgeteilt werden.
Die Registereinheit wird meistens wiederum in 2- bis 4-bit-Elemente (Slices) getrennt. Damit wird es möglich, mit den gleichen Grundelementen Prozessoren mit verschiedenen Wortlängen aufzubauen.

Die *Mikroprogrammsteuereinheit* steuert das Mikroprogramm, wobei der Operationsteil der (aus dem Arbeitsspeicher geholten) Instruktion die Startadresse des Mikroprogramms erzeugt (siehe Kap. 3).

Der *Mikroprogrammspeicher* (ROM) enthält die Mikroinstruktionen (z. B. 100 × 23 bit beim GPC/P-System von National Semiconductor, 512 × 24 bit beim AMI-7200-System von American Micro Systems).

Bei den verschiedenen Produkten werden sehr unterschiedliche Chip-Aufteilungen vorgenommen. Die folgende Darstellung zeigt die bei den verschiedenen Herstellern verwendete Aufteilung.

Tabelle 3 Übersicht der Organisation beziehungsweise Chip-Aufteilung von Multichip-Prozessoren. Falls für den Mikroprogrammspeicher keine Standardspeicherelemente verwendet werden können, ist zur Entwicklung des Mikroprogramms ein diskret aufgebauter Simulator erforderlich.

Hersteller	Mikroprogramm Steuereinheit	Mikroprogramm Speicher	Registereinheit und ALU	Bemerkungen
American Microsystem AMI 7200				Entwicklung dieses Mikroprozessors eingestellt
American Microsystem AMI 7300				dito
INTEL 3000		Standard ROM oder PROM	2 bit Slices	
National Semiconductor GPC P			4 bit Slices	
Monolithic Memories 5701 6701	diskrete Elemente	Standard ROM oder PROM	4 bit Slices	
Texas Instruments SBP 0400	diskrete Elemente	Standard ROM oder PROM	4 bit Slices	

5. Wortlänge von Mikroprozessoren

Unter Wortlänge wird die Anzahl bit der Operanden (Daten) verstanden. In einigen Mikroprozessoren sind verschiedene Operandenwortlängen möglich (z. B. INTEL MCS 8080: hauptsächlich 8-bit-Operanden, einige 16-bit-Operanden für Inkrement-, Dekrement-, Exchange- und Addier-Instruktionen).
Die Operandenwortlänge ist oft nicht identisch mit der Wortlänge von Instruktionen und Adressen.
Die folgende Tabelle zeigt die verschiedenen gebräuchlichsten Wortlängen von Mikroprozessoren, wobei zurzeit 8-bit-Prozessoren die grösste Bedeutung (hinsichtlich Anzahl erhältlicher und angekündigter Typen) aufweisen.

Tabelle 4 Übersicht der bei Mikroprozessoren üblichen Wortlängen und Adressierbereiche

Operandenwortlänge in bit	Instruktionswortlänge in bit	Adressbereich Daten/Instr.	Beispiele
4	8...20	1 K...4 K (4 K...16 K)	MCS 4004, MCS 4040 PPS 4, IMP 4
8	8...24	16 K...65 K	MCS 8008, MCS 8080 MC 6800, PIP, PPS 8
16	16	65 K	IMP 16, CP 1600, PACE

6. Synchrone, asynchrone und pseudoasynchrone Arbeitsweise

Synchrone Mikroprozessoren arbeiten mit einer konstanten Geschwindigkeit, welche bei der Inbetriebnahme durch die Taktfrequenz bestimmt ist (z. B. MC 6800, IMP 16).
Asynchrone Mikroprozessoren arbeiten hinsichtlich externem Datenverkehr (Speicher, Ein-/Ausgabe-Einheiten) in einem «Handshakingverfahren», das heisst, die Geschwindigkeit für den Ablauf des Datentransfers wird von einem Quittierungssignal (Ready line beim MCS 8080) des Speichers (nach Ablauf derZugriffszeit) oder der Ein-/Ausgabe-Einheit beeinflusst.
Damit ist es möglich, an einem Mikroprozessor Speicher verschiedener Zugriffszeit optimal zu verwenden (z. B. schnelle dynamische RAMs mit langsamen PROMs).
Ein asynchroner Mikroprozessor kann gleich wie ein synchroner Mikroprozessor arbeiten, falls das Quittierungssignal an einen konstanten logischen Pegel gelegt wird; dann arbeitet der Computer konstant mit der – von der Taktfrequenz abhängigen – maximalen Geschwindigkeit. Das Quittierungssignal kann auch zum Austesten von Programmen zum Ausführen von Einzelzyklen (Single Machine Cycles), das heisst zum Anhalten des Computers nach Ausführung von 1 Zyklus verwendet werden.

Pseudoasynchrone Mikroprozessoren:
Synchrone Mikroprozessoren können bis zu einem gewissen Grade durch Anwendung von «phase stretching» auch asynchron arbeiten.
Bei dieser Methode wird die Taktfrequenz, beziehungsweise -periode des Mikroprozessors durch eine externe Beschaltung von der Zugriffszeit von Speichern und Ein-/Ausgabe-Interfaces beeinflusst, damit wird es möglich, mit kurzzeitigem «Anhalten» des Taktgenerators eine externe Geschwindigkeitsanpassung zu erreichen.

Auch die *asynchronen* Mikroprozessoren arbeiten meist nicht exakt asynchron (im Sinne der in der Digitaltechnik üblichen Definition), sondern sind via Taktsignal synchronisiert, das heisst, die Geschwindigkeitsanpassung (Bremsen!) erfolgt in ganzzahligen Zeitinkrementen (z. B. $\approx n \times 500$ ns beim INTEL MCS 8080).

7. Parallele und zeitmultiplexe Datenübertragung

In einem Instruktionszyklus wird zu Beginn von der Zentraleinheit die Adresse der betreffenden Instruktion an den Programmspeicher übertragen, anschliessend erfolgt der Transfer der Instruktion in umgekehrter Richtung.
Bei speicherbezogenen Instruktionen (memory reference instructions) findet im nächsten Zyklus durch Ausgabe der Operandenadresse und Transfer des Operanden nochmals ein analoger Vorgang statt.
Tabelle 5 zeigt die bei den erhältlichen Mikroprozessoren angewandten Übertragungsprinzipien. Methode 2 unterscheidet sich von Methode 3 nur dadurch, dass der gemeinsame Bus für Adressen und Daten für eine parallele Übertragung der Adresse nicht ausreicht, beziehungsweise die Adresse in zwei aufeinanderfolgenden Paketen übertragen werden muss.
Die Mikrocomputerentwicklung tendiert zurzeit eindeutig nach Methode 4, das heisst parallele und damit simultane Adress- und Datenübertragung über separate Adress- und Datenkanäle mit Verwendung von 40-Stift-Gehäusen.
Beim PPS-8 wird zur Erhöhung der Geschwindigkeit des PMOS-Elementes überlappende Adressierung von ROM (Instruktionen) und RAM (Operanden) verwendet, das heisst, während des Anlegens der ROM-Adresse stehen die RAM-Daten der vorherigen Instruktion am Datenbus, während des darauffolgenden Anlegens der RAM-Adresse steht der Inhalt der vorher adressierten ROM-Zelle auf dem Datenbus. Voraussetzung dieser Methode sind Adressregister in den Speichern.

8. Register und Flags

Im folgenden werden nur jene Register und Flags betrachtet, welche durch Instruktionen adressierbar sind. In jedem Computer sind zusätzliche Register und Flags für Zwischenspeicherung, Ausgabebuffer usw. vorhanden, welche der Programmkontrolle nicht zugänglich sind.
Die ersten beiden Register sind notwendige Voraussetzungen für einen Prozessor, die weiteren Register variieren von Hersteller zu Hersteller. Eine grosse Registerzahl zusammen mit einem externen Stack erleichtert die «Reentrant»-Programmierung (Programme, welche unterbrochen und von anderen Stellen wieder aufgerufen werden können, ohne dass die variablen Grössen des unterbrochenen Programms «verloren» gehen).

1. Programmzähler: enthält die Adresse der laufenden Instruktion (12...16 bit)

2. Akkumulator: kann als Zwischenspeicher für Datentransfer zwischen Speicher und externen Geräten (Input/Output) sowie zum Abspeichern von Operanden und Resultaten verwendet werden.
Bei arithmetischen Operationen wird der Operand im Akkumulator durch das Resultat überschrieben.
Die Wortlänge des Akkumulators entspricht der Operandenwortlänge (4...16 bit). Einige Mikrocomputer (z. B. MC 6800, IMP 4, IMP 8, IMP 16) enthalten mehrere Akkumulatoren.

3. Adressregister, Indexregister, Scratchpadmemory: Adressregister

Tabelle 5 Parallele und zeitmultiplexe Datenübertragung von Adressen, Instruktionen und Operanden bei Mikroprozessoren. Zur Erhöhung der Geschwindigkeit und Reduktion des extern erforderlichen Aufwands werden bei modernen Mikroprozessoren separate Adress- und Datenkanäle verwendet.

Methode	Adress- und Datenbus		Zeitmultiplexe Übertragung von		Anzahl Stifte der CPU	Beispiele	Bemerkungen	Trends
	gemeinsam	separat	Adresse	Daten				
1. Zeitmultiplexe Übertragung von Adressen, Instruktionen und Operanden in 4-bit-Paketen über gemeinsamen 4-bit-Bus	4-bit bidirectional		3×4 bit	2×4 bit	16 24	MCS 4004 MCS 4040	grosser zusätzlicher Logikaufwand für Interfacing von Standardspeichern	↑ zunehmender Aufwand für externe Logik (Adress-, Operandenregister usw.) ↓ zunehmende Geschwindigkeit ↓ zunehmende Stift-Zahl der CPU
2. Zeitmultiplexe Übertragung von Adressen und Instruktionen bzw. Operanden in 8-bit-Paketen über gemeinsamen 8-bit-Bus	8-bit bidirectional		1×8 bit gefolgt von 1×6 bit	1×8 bit	18	MCS 8008	externes Adressregister erforderlich	
3. Zeitmultiplexe Übertragung von Adressen und Instruktionen, bzw. Operanden über gemeinsamen 16-bit-Bus	16-bit ¹) bidirectional		1×16 bit	1×8 bit bzw. 1×16 bit	²) 40	GPC/P IMP 8 IMP 16 MK 5065	externes Adressregister erforderlich	
4. Parallele Übertragung von Adressen und Instruktionen, bzw. Operanden über separaten Adress- und Datenbus		Adressbus 12...16 bit unidirectional, Datenbus 8 bit bidirectional	parallel 12...16 bit für Adressen, 8 bit für Daten		40... 42	PPS 4 PPS 8 PIP MCS 8080 MC 6800		

¹) Bei IMP 8 und IMP 16 erfolgt eine Aufteilung des an der RALU gemeinsamen Adress- und Datenkanals in separate Kanäle an den Printplattenanschlüssen
²) IMP 8 und IMP 16 sind Multichipprozessoren

und Indexregister dienen zum Adressieren von Operanden, wobei bei Indexregistern die Adresse des Operanden aus der Summe des Registerinhaltes und dem Adressteil (Offset) der Instruktion gebildet wird, ohne dass dabei eine Änderung des Indexregisterinhaltes erfolgt.
Für Adress- und Indexregister werden bei Mikroprozessoren oft mehrere einzeln adressierbare Register (z. B. H- und L-Register beim MCS 8008 und MCS 8080) zusammengenommen.
Mit Scratchpadmemory werden Register bezeichnet, welche zum Zwischenspeichern von Operanden, Resultaten usw. verwendet werden (Notizblock).
Meistens sind bei diesen Registern nur Transfer-, Inkrement-, Dekrement-, (evtl. Addier-)manipulationen möglich; oft ist die Beeinflussung der Control-Flags (Carry-, Sign usw.) sehr eingeschränkt.

4. Stackpointer (s. Kap. 9)

5. Control-Flags sind einzelne Flip-Flops, deren Zustand durch die Resultate arithmetisch logischer Operationen oder beim Datentransfer beeinflusst werden. Die Zustände dieser Flags können via bedingte Sprungfunktionen abgefragt werden. Die Beeinflussungen sind bei den verschiedenen Mikroprozessoren sehr unterschiedlich und teilweise schlecht überschaubar. Bei einigen Mikroprozessoren werden die Control-Flags bei Transfer- und Doppeloperandeninstruktionen (Dekrementieren, Inkrementieren, Addieren) nicht beeinflusst (MCS 8008, MCS 8080) oder bei einigen Instruktionen nur einzelne (nicht alle) Control-Flags gesetzt. Einige Flags sind durch spezielle Instruktionen wie «Set Carry», «Complement Carry» beeinflussbar.

Bei Mikroprozessoren sind folgende Flags üblich:

- Carry, Übertrag vom MSB (most significant bit),
- Zero, wird bei Inhalt Ø gesetzt,
- Sign, entspricht MSB (most significant bit),
- Parity, wird gesetzt, wenn die Anzahl 1-Bits gerade (ungerade) ist.
- Overflow, wird gesetzt bei arithmetischem Überlauf. Da beim Rechnen mit positiven und negativen Zahlen das MSB zur Kennzeichnung des Vorzeichens verwendet wird (1 ≙ negativ), ist es beispielsweise möglich, dass bei der Addition zweier positiver Zahlen ein negatives Resultat entsteht.

Der Overflow-Flag wird somit immer gesetzt, wenn der Zahlenbereich (positiv und negativ) überschritten wird; er entspricht einer «Exklusiv-ODER-Verknüpfung von Sign- und Carry-Flag.

- Hilfscarry, wird bei 8-bit-Mikroprozessoren beim Überlauf von Bit

3 gesetzt. Er ist bei jenen Mikroprozessoren vorhanden, welche BCD(Binär Dezimal)-Operationen aufweisen.
- Stackoverflow, wird beim Überlauf vom internen Stack (z.B. IMP 8, IMP 16) gesetzt.
- Vom Programm, das heisst durch Instruktion setzbare Flags: Einige Mikroprozessoren enthalten Flags, welche von Instruktionen direkt gesetzt (z.B. zum «Merken» eines bestimmten Zustandes) und via bedingte Sprungfunktionen abgefragt werden können.
- Ein weiterer Flag dient zum Sperren des Interrupt-Eingangs (Interrupt Enable/Disable Flag). Dieser kann in der Regel jedoch nicht via bedingte Sprungfunktionen getestet werden.

Die Anzahl und Art der Flags und vor allem die Art der Beeinflussung der Flags aus dem Programmablauf ist ein sehr wesentlicher – meist unterschätzter – Faktor bei der Beurteilung des Instruktionssatzes.

9. Stack (Stapel, LIFO, Last in first out)

9.1 Begriff und Verwendung

Unter dem Begriff Stack wird eine lineare Liste verstanden, bei der Zugriffe (Insertions, Deletions) nur an einem Ende dieser Liste erfolgen können (s. Bild 11 a). Für das Abspeichern auf den Stack wird der Begriff «PUSH» und für den Zugriff beim Lesen «POP» verwendet (Bild 10). Für die Zentraleinheit eines Mikrocomputers erfüllt der Stack folgende Funktionen:

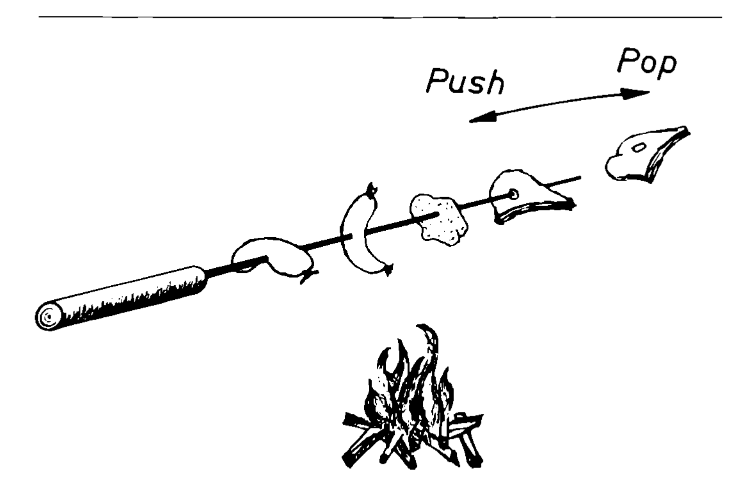


Bild 10 Veranschaulichung des «Push down Stacks».
Ein Zugriff auf den Stack kann immer nur über die Spitze erfolgen.
Die dem Aufstecken entsprechende Funktion wird mit «Push» bezeichnet, für das Wegnehmen gilt der Ausdruck «Pop».

1. *Abspeichern der Rücksprungadressen* beim Aufruf von *Subroutinen.*
Subroutinen sind Unterprogramme, welche von beliebigen Stellen eines Hauptprogramms aufgerufen werden können. Nach Ablauf der Subroutine wird an die der aufrufenden Stelle folgende Instruktion zurückgesprungen. Deshalb ist es notwendig, bei jedem Aufruf die Rücksprungadresse zwischenzuspeichern. Da innerhalb einer Subroutine wiederum neue Subroutinen aufgerufen werden können (Verschachtelung, Nesting), wird das Abspeichern von mehreren Rücksprungadressen erforderlich, welche gemäss dem Stackprinzip (LIFO) zugreifbar sein müssen.
2. Bei *Interrupts* wird der Programmzähler hardwaregesteuert auf einen vorgegebenen Wert (Interruptvektor) gesetzt; bei dieser Adresse startet das entsprechende Interrupt-Service-Programm. Da sich im Zeitpunkt des Interrupts in den Registern und Flags aktuelle Werte befinden, welche nach Ausführung der Interrupt-Service-Routine wieder für die Weiterabarbeitung des Programms benötigt werden, muss der Inhalt der von der Interrupt-Service-Routine verwendeten Register und Flags vor ihrer Benutzung abgespeichert werden.
Somit wird die Interrupt-Service-Routine am Anfang vorerst die aktuellen Werte der Register und Flags abspeichern und vor dem Rücksprung in das unterbrochene Hauptprogramm wieder in die Register und Flags laden.
Falls Interrupt-Service-Routinen ebenfalls wiederum durch Interrupts unterbrochen werden können *(Multiple Interrupt-handling)* ist für das Abspeichern von Registern und Flags (Computerstatus) ebenfalls ein Stack erforderlich.
3. Stacks können auch von den Programmen zum Zwischenspeichern von Operanden, Zwischenresultaten usw. oder zur Übergabe von Parametern verwendet werden.

9.2 Implementationen von Stacks bei Mikroprozessoren

Man unterscheidet zwischen internen und externen Stacks. *Interne Stacks* befinden sich innerhalb der Zentraleinheit und sind deshalb in der Grösse stark begrenzt (4 bis 16 Levels, das heisst Abspeicherkapazität von 4 bis 16 Rücksprungadressen oder entsprechendes Speichervolumen für Register-, Flags-Abspeicherung usw.).

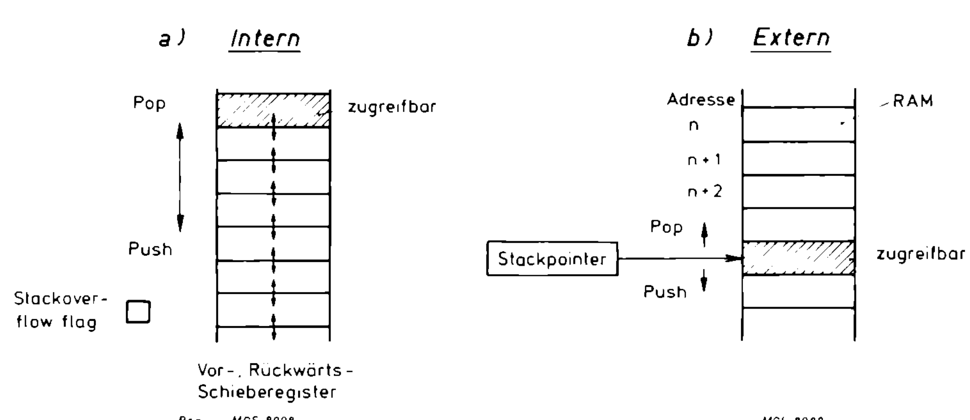


Bild 11 Prinzip des internen und externen Stacks.
Interne Stacks arbeiten oft auf der Grundlage von parallel geschalteten Vor/Rückwärts-Schieberegistern.
Externe Stacks werden in konventionellen Speichern (RAMs) implementiert, wobei die Adresse durch einen im Prozessor befindlichen Stackpointer gebildet wird.

Externe Stacks befinden sich im Arbeitsspeicher des Computers. Der Zugriff zum Stack wird über den in der Zentraleinheit befindlichen Stackpointer adressiert (s. Bild 11 b). Der Stackpointer wird bei jedem Zugriff automatisch dekrementiert (PUSH) beziehungsweise inkrementiert (POP).

Vergleich von internen und externen Stacks
Durch die begrenzte Grösse der internen Stacks ist die Subroutinenverschachtelung und vor allem das Interrupt-handling (Multiple Interrupt-handling) stark eingeschränkt.
Bei einigen internen Stacks (z. B. MCS 8008) können nur Subroutinenrücksprungadressen (d. h. Programmzähler) abgespeichert werden, jedoch keine allgemein verwendbare Register und Flags. Dies verunmöglicht praktisch ein Arbeiten mit mehreren Interrupts beziehungsweise Interruptverursachern, da die aktuellen Werte in der Zentraleinheit (CPU-Status) nicht auf den Stack gerettet werden können. Ferner stellt sich bei Verwendung von Kernspeichern oder batteriegepufferten Halbleiterspeichern (CMOS-RAMs, dynamische MOS-RAMs) oft das Problem des Abspeicherns der aktuellen Werte (CPU-Status) in den (nonvolatilen) Arbeitsspeicher. Nicht alle internen Stacks (z. B. MCS 8008) sind nach aussen transferierbar. Bei einigen Mikroprozessoren ist der Stackinhalt zwar transferierbar, es wird jedoch relativ lange Zeit für den Transfer benötigt.

9.3 Mikroprozessoren mit mehreren Registersätzen (MK 5065 P)

Der Mikroprozessor MK 5065 P von Mostek ist nicht in die vorher diskutierten Kategorien einteilbar. Beim MK 5065 P sind drei voneinander unabhängige Registersätze (3 Programmzähler, 3 Akkumulatoren) und ein externer Stack für die Abspeicherung von Rücksprungadressen verwendet.
Bei einem Interrupt wird einfach von einem Registersatz (Level 1) auf den andern Registersatz (Level 2) umgeschaltet. Die Interrupt-Antwortzeit ist somit nur von der Dauer der laufenden Instruktion abhängig, da die Umschaltung praktisch zeitlos erfolgt. Hingegen ist die Anzahl Unterbrechungsstufen auf zwei begrenzt.
Eine abgeschwächte Form dieser Methode ist auch beim MCS 4040 enthalten, der einen umschaltbaren 8×4-bit-Registersatz enthält.

9.4 Probleme bei Anwendung von ROMs bei Computern, welche die Rücksprungadresse am Anfang der Subroutine abspeichern

Bei älteren Minicomputern (z. B. PDP 8) wird beim Aufruf einer Subroutine (Bild 12), anstelle der Verwendung von Stacks, stets der Inhalt des Programmzählers in der ersten Zelle der Subroutine abgespeichert. Das Abspeichern erfolgt beim Ausführen der JSR-Instruktion (Jump to Subroutine) vollkommen Hardware-gesteuert; die erste Zelle wird von der Subroutine selbst nicht verwendet. Der Rücksprung in das Hauptprogramm geschieht mit einer indirekten Sprunginstruktion über die erste Adresse der Subroutine.

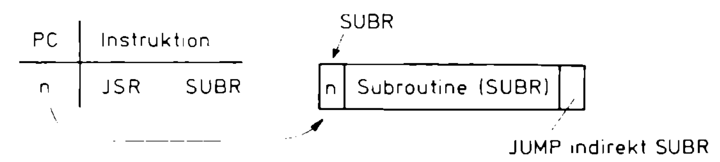


Bild 12 Abspeicherung der Rücksprungadresse bei Verwendung von Subroutinen am Subroutinenanfang (z. B. PDP 8).

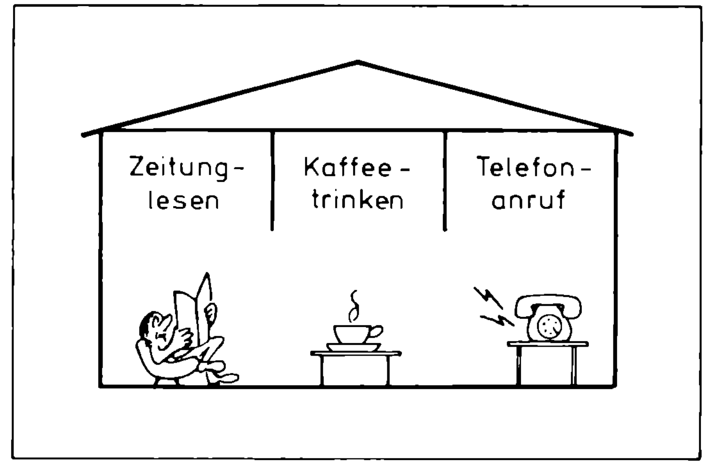


Bild 13 Veranschaulichung von Prioritäten und Interrupts im täglichen Leben. Erfolgt beim Lesen einer Zeitung («normales» Programm) ein Telefonanruf (Interrupt), so wird das Lesen unterbrochen. Dabei soll die Textstelle vermerkt werden (Rücksprungadresse), an der nach dem Telefongespräch (Interrupt Service-Routine) weitergelesen wird.
Vielleicht wird während des Telefongesprächs jemand an die Tür klopfen, womit eine weitere Unterbrechung erforderlich wird (Multiple-Interrupt-Handling).

Da bei Mikrocomputern die Programme meistens in ROMs oder PROMs abgespeichert sind, ist dieses Verfahren nicht anwendbar.
Beim PDP 8 A wurden deshalb aus diesen Gründen durch Verwendung eines zusätzlichen Bits (13. Bit) besondere Massnahmen erforderlich. Ebenfalls beim PDP 8-kompatiblen IM 6100 von INTERSIL werden diesbezügliche Probleme entstehen.

10. Interrupts (Bild 13)

10.1 Prinzip

Das Interruptsignal am Eingang der Zentraleinheit erlaubt das Unterbrechen eines Programms nach Ablauf der laufenden Instruktion. Die Hardware der Zentraleinheit erzwingt – nach Aktivieren des Interruptsignals – die Ausführung einer «Call to Subroutine»-Instruktion.
Dadurch wird:
1. Der aktuelle Programmzähler an einer vorgegebenen Stelle (feste Speicherzelle, interner oder externer Stack) gerettet.
2. Der Programmzähler wird mit einem neuen Wert geladen, wobei in den meisten Fällen eine feste Zuordnung zwischen einem Interrupt-Eingang (Interrupt-Level) und diesem neuen Wert (Interrupt-Vektor, Trap Location) besteht. Der Interrupt-Vektor bildet die Startadresse für die Interrupt-Service-Routine.
3. Wie bereits in Kapitel 9 erwähnt, muss von der Interrupt-Service-Routine auch der aktuelle Zustand des Computers (Register, Flags) gerettet werden.
Es sollte die Möglichkeit bestehen, den Interrupt-Eingang unter Programmkontrolle zu sperren (EI/DI, Enable-Interrupt/Disable-Instruktion). In der Regel wird der Interrupt-Eingang automatisch nach dem Akzeptieren eines Interrupts gesperrt, dies verhindert das mehrfache Ausführen einer Interrupt-Sequenz durch die gleiche Interrupt-Ursache (beim MCS 8008 nicht vorhanden).
Die externe Hardware, welche den Interrupt anlegt, muss über das Akzeptieren des Interrupts orientiert werden, damit das Interrupt-Signal wiederum in den inaktiven Zustand gebracht wird; dies kann durch die Hardware der Zentraleinheit unmittelbar während des Akzeptierens des Interrupts erfolgen (Interrupt Acknowledge-Signal) oder via Software durch Statusabfrage, Kontrollwortausgabe oder Datentransfer.

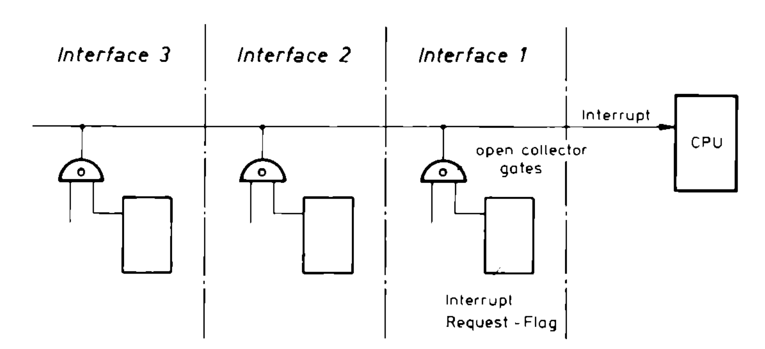


Bild 14 Mehrere Interrupt Verursacher an einer gemeinsamen Interrupt Leitung. Lokalisierung der Interrupt Ursache erfolgt durch Polling (Abfragen).

10.2 Multilevel-Interrupt-Systeme

- Mehrere Interrupt-Verursacher an einer Interrupt-Leitung:
Die Interrupt-Request-Flags verschiedener Interfaces sind via Open-Kollektor-Tore mit einer gemeinsamen Interrupt-Leitung verbunden (Bild 14).
Um den Interrupt-Verursacher zu lokalisieren, sind folgende Methoden verwendbar:

1. Programmgesteuertes sequentielles Abfragen der Interrupt-Request-Flags von jedem Interface.
2. Die Interrupt-Request-Flags (z.B. maximal 8 Flags bei einem 8-bit-Mikroprozessor) werden mit einer einzigen Abfrage in den Akkumulator transferiert. Es besteht die Möglichkeit, die Interrupt-Request-Flags unter einer gemeinsamen Device-Adresse oder mit einer speziellen Steuerleitung (Interrupt-Statusrequest) abzufragen. Die Lokalisierung der einzelnen Bits erfolgt programmgesteuert.
3. Siehe 2, Abfrage und Lokalisierung erfolgen mikroprogrammgesteuert (s. Kap. 3.2).
4. Prioritätskette
Bild 15 zeigt das Prinzip der Prioritätskette (wired chain priority-propagation).

Das Anlegen der Interrupts an die gemeinsame Interrupt-Leitung erfolgt analog der in Bild 14 gezeigten Methode. Beim Akzeptieren des Interrupts wird vom Computer ein Interrupt-Grant-Signal erzeugt, welches sich über die Prioritätskette bis zu jenem Interrupt-verlangenden Interface mit der höchsten Priorität fortpflanzt. Nur dieses Interface antwortet durch Anlegen seines Interrupt-Vektors V (z. B. 8-bit-Adresse) an den Datenbus. Damit springt das Programm an die dem Gerät zugeordnete Interrupt-Service-Routine. Dieser Ablauf kann sowohl programm- als auch mikroprogrammgesteuert werden. Als Nachteil dieses Konzepts muss erwähnt werden, dass die Priorität durch die örtliche Reihenfolge der einzelnen Interfaces gegeben ist und dass beim Herausziehen einer Interfaceplatte das verbleibende System nicht mehr funktioniert.

Mehrere Interrupt-Eingänge an der Zentraleinheit:
Einige Mikroprozessoren besitzen mehrere Interrupt-Eingänge (PPS 8, MC 6800 usw.), wobei einzelne maskierbar (sperrbar) sind und andere nicht.
Bild 16 zeigt das Prinzip einer externen Interrupt-Logik zum Mikroprozessor MCS 8080:
Der MCS 8080 weist selbst nur einen Interrupt-Eingang auf. Durch eine externe Logik ist es möglich, das System auf 8 Interrupt-Eingänge auszubauen. Dabei wird die ODER-Funktion aller Interrupt-Levels mit dem Interrupt-Eingang des Mikroprozessors verbunden.
Das Akzeptieren des Interrupts wird vom Computer durch Ausgabe des Statusbits INTA gekennzeichnet, gleichzeitig wird ein leerer Zyklus eingefügt (d. h. der Programmzähler wird nicht erhöht). Dieser leere Zyklus wird von der externen Logik dazu benützt, via Prioritäts-Encoder eine Restart-Instruktion an den Datenbus anzulegen. Die Restart-Instruktion entspricht einer «Call to Subroutine»-Instruktion, sie erlaubt jedoch nur, an 8 verschiedene Adressen zu springen.
Der Prioritäts-Encoder erzeugt an seinem Ausgang den Vektor (0...7), der dem Eingang (0...7) mit der höchsten Priorität entspricht.
Neue Bausteine zur Interruptbehandlung ermöglichen das Sperren von Interrupteingängen, welche eine durch Software veränderbare Priorität nicht erreichen (INTEL 8214). Ferner werden Bausteine angeboten, welche eine zyklische Zuordnung der Prioritäten erlauben und mit denen Interrupteingänge individuell gesperrt werden können (INTEL 8259).

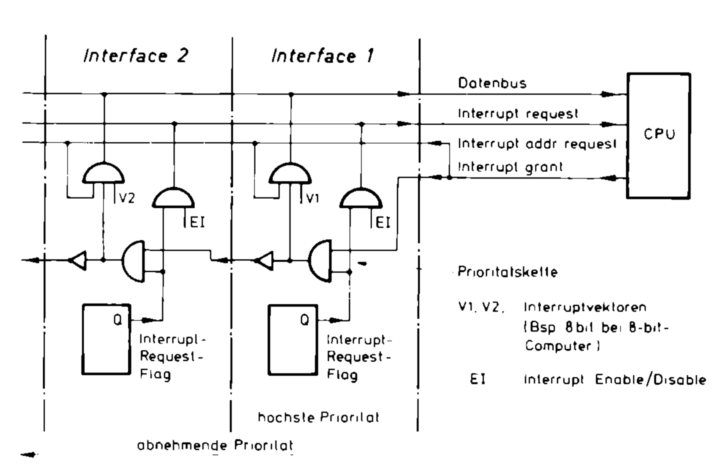


Bild 15 Mehrere Interrupt Verursacher an einer gemeinsamen Interrupt Leitung mit Prioritätskette und Vektorübergabe. Das am nächsten beim Prozessor befindliche Interface hat die höchste Priorität. Das via Interrupt-Grant Leitung ausgewählte Interface überträgt seine Vektoradresse an den Prozessor.

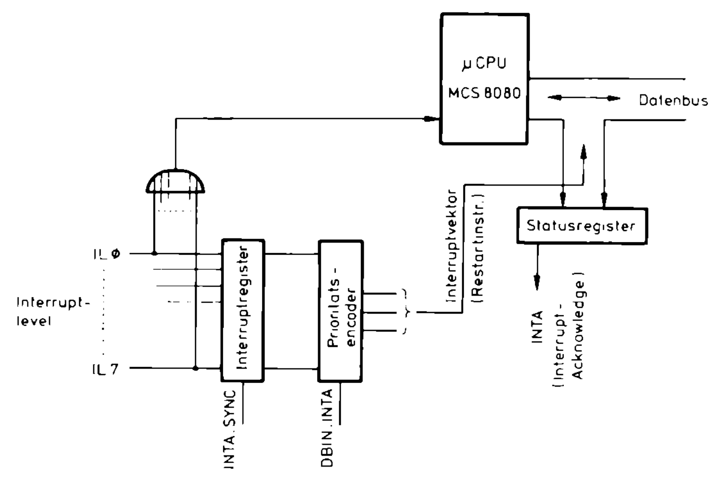


Bild 16 Externe Interrupt Logik beim Mikroprozessor MCS 8080. Während des Akzeptierens eines Interrupts (INTA) wird von der Interrupt-Logik eine RESTART-n Instruktion an den Datenbus angelegt. Dies bewirkt einen Programmsprung nach Adresse n, wobei der laufende Programmzähler auf den Stack gerettet wird. Der Prioritätsdecoder erzeugt den Adressteil n innerhalb der RESTART Instruktion und setzt die Prioritäten beim gleichzeitigen Anliegen von mehreren Interrupts.

10.3 Reset

Natürlich kann ein Programm auch mit Betätigung der Reset-Leitung unterbrochen und an eine vorgegebene Adresse gebracht werden. Im Unterschied zum Interrupt geht der Inhalt des laufenden Programmzählers jedoch verloren.

10.4 Software-Interrupt

Ein Software-Interrupt erzeugt die gleiche Funktion wie ein Hardware-Interrupt, das heisst, der Programmzähler wird gerettet, der Interrupt-Eingang gesperrt und ein neuer Programmzählerinhalt (Trap-Vektor) geladen. Die Software-Interrupt-Instruktion (SWI bei MC 6800) entspricht einer Einwort-«Call to Subroutine»-Instruktion mit zusätzlicher Interrupt-Disable-Funktion und evtl. Retten der Register auf den Stack.

11. Input/Output-Architektur

Man unterscheidet zwischen programmgesteuertem Input/Output und direktem Speicherzugriff (DMA, Direct Memory Access).

Beim *direkten Speicherzugriff* wird die Zentraleinheit während des Datentransfers funktionell abgeschaltet, das heisst, die Zentraleinheit wird in einen Wartezustand gezwungen, dabei werden die Adress- und Datenleitungsausgänge der Zentraleinheit hochohmig. Das externe Gerät, das den DMA verlangt, verbindet sich nun über den Adress- und Datenbus mit dem Speicher und erzeugt auch das Timing für den Datentransfer, das Inkrementieren der Adresse usw. Werden mehrere Geräte mit DMA betrieben, so wird eine entsprechende DMA-Prioritätssteuerung notwendig. Beim PPS 8 und MCS 8080 sind integrierte DMA-Controller (DMAC) erhältlich.

Beim *programmgesteuerten* Datentransfer sind folgende drei Systeme zu unterscheiden:

1. Input/Output wird gleich wie der Datentransfer in den Speicher behandelt, das heisst, Adressierung, Timing und Instruktionen sind identisch mit einem Speicherzugriff. Damit ist die Anzahl der adressierbaren «Devices» praktisch unbegrenzt (z. B. MC 6800, LSI 11).

2. Separate Instruktionen, Statussignale usw. für Input/Output. Bei den meisten Mikroprozessoren ist die Anzahl der adressierbaren «Devices» begrenzt (z. B. MCS 8008: 8 Input, 24 Output; MCS 8080, PIP: 256 In-/Output).

3. Verwendung von I/O-Ports:
Hier wird kein Buskonzept verwendet, das Daten und Adressleitungen enthält, sondern diskrete I/O-Ports.
Beispielsweise kann der Input/Output-Transfer beim MCS 4004 und MCS 4040 via Input/Output-Ports erfolgen; diese befinden sich an den Ausgängen der Speicher (RAM 4002, ROM 4001, ROM 4308). Beim PPS 4 sind an den Pins der Zentraleinheit diskrete I/O-Ports vorhanden.

Natürlich kann bei einem Mikroprozessor, dessen Input/Output nach dem 2. Verfahren arbeitet, vom Anwender das 1. Verfahren verwendet werden. Bei einigen Anwendungen ergeben sich daraus Vorteile.
Hinsichtlich synchronen und asynchronen Datentransfers wird auf Kapitel 6 verwiesen.

12. Instruktionsset
12.1 Allgemeines

Bei der Beurteilung eines Mikroprozessors ist – neben der Architektur – das Instruktionsset, das diese Architektur unterstützt, von gleichwertiger Bedeutung.
Aussagefähige Bewertungen des Instruktionssets von Computern sind vor allem mittels «Benchmarks» möglich. Bei dieser Methode werden typische Teilaufgaben einer gegebenen Anwendung mit den Instruktionen verschiedener Computer programmiert und der Speicherbedarf sowie die Ablaufzeit verglichen.
Die vom Hersteller angegebene Anzahl der Instruktionen ist sehr wenig aussagefähig.
Die Leistungsfähigkeit eines Instruktionssets kann davon beeinflusst werden, ob als Programmspeicher ROMs oder RAMs verwendet werden.
Nachstehend werden anhand der verschiedenen Funktionen des Instruktionssets die wichtigsten Bewertungsparameter dargestellt.
Die folgende Einteilung beruht auf Funktionsklassen, wobei eine Instruktion durchaus zwei (oder drei) Funktionen erfüllen kann. Beispielsweise erfüllt die Instruktion ADD M (Addiere Inhalt der Speicherzelle M) sowohl eine arithmetische als auch eine Transferfunktion. Die Einteilung nach Funktionen wurde gewählt, weil diese eher einen Vergleich zwischen den Instruktionsrepertoires verschiedener Mikroprozessoren ermöglicht.

12.2 Funktionen des Instruktionssets
12.2.1 Arithmetisch/logische Funktionen:

+ Add, Add with carry
- Subtract, Subtract with borrow
∧ AND
∨ OR
⊻ Exklusiv OR
  Compare
  Increment, Decrement
  Shift, Rotate
  Complement
  Decimal Adjust

Bei den meisten Mikroprozessoren sind annähernd alle der obigen Funktionen enthalten, lediglich beim MCS 4004 sind keine logischen Operationen vorhanden. Der IMP 16 enthält im erweiterten Instruktionsset Multiply und Divide von 16-bit-Operanden. Add with carry und Subtract with borrow werden für «multiprecision-arithmetik», das heisst, bei Addition und Subtraktionen mit mehrfacher Wortgenauigkeit benötigt. Ein wesentlicher Bewertungsfaktor ist die Beeinflussung der Flags durch die obenstehenden Operationen. Add-, Increment- und Decrement-Operationen (Double precision) von Registerpaaren (z. B. MCS 8080) verlieren an Bedeutung, wenn keine Flags beeinflusst werden, da man allfällige Überträge, Nullwerte usw. nur über zusätzliche Compare-Operationen feststellen kann.
Bei Add- und Subtract-Operationen mit positiven und negativen Operanden, ist das Vorhandensein eines arithmetischen Overflow-Flags (z. B. MC 6800, IMP 8, IMP 16) von Bedeutung.

12.2.2 Transferfunktionen

Bei Transferfunktionen wird ein Operand von einer Stelle (Source [z. B. Register, Speicherzelle]) an eine andere Stelle (Destination) verschoben. Dabei wird der frühere Inhalt der Destination überschrieben,

der Inhalt der Source bleibt unverändert. Wesentliches Bewertungskriterium hinsichtlich Transferfunktionen von Mikroprozessoren bildet die Adressierbarkeit von Source und Destination. Die auf dem Markt angebotenen Mikroprozessoren unterscheiden sich hinsichtlich der vorhandenen Adressiermöglichkeiten sehr stark.

Es werden folgende Adressiermodes unterschieden:

a) Registeradressierung (implied addressing)
Mit dieser Adressierungsart kann auf die verschiedenen internen Register der Zentraleinheit (Akku, Indexregister, Stackpointer usw.) zugegriffen werden. Die Adressierung der Register ist bei jedem Computer vorhanden; bei mindestens einem Hersteller wird diese Methode jedoch in den Manuals speziell erwähnt (Implied addressing, MC 6800).

b) Absolute Adressierung (direct addressing)
Die Adresse des Operanden ist in der Instruktion enthalten, z. B. LDA 250 = Lade den Akku mit dem Inhalt der Speicherzelle 250.
Bei 8-bit-Prozessoren benötigt die absolute Adressierung eine 3-Byte-Instruktion und ist somit sehr speicherintensiv. Falls sich die Programme in ROMs befinden, ist die Adresse nicht unter Programmkontrolle variierbar.
– *Beschränkte absolute Adressierung* (abbreviated addressing). Einige Mikroprozessoren (z. B. MK 5056 P, IMP 8) weisen die Möglichkeit einer beschränkten absoluten Adressierung von «page Ø» und der «current page» auf. Eine «page» entspricht einem Speicherbereich von 256 Bytes, wobei page Ø den Adressbereich Ø...255 und current page den Adressbereich $n \cdot 256 \ldots [(n+1) \cdot (256-1)]$ – innerhalb der sich der Programmzähler befindet – umfassst. Es handelt sich um 2 Byte-Instruktionen (1. Byte ≙ Operationscode, 2. Byte ≙ Adresse innerhalb page).
Die «Current page»-Adressierung hat bei Anwendungen von ROMs (für die Programmspeicherung) nur einen beschränkten Wert, da lediglich zu konstanten, jedoch nicht zu variablen Operanden zugegriffen werden kann (ist somit beispielsweise durch immediate adressing ersetzbar).
Beim MC 6800 ist ausser der absoluten Adressierung (3-Byte-Instruktion) eine Adressierung von page Ø (2-Byte-Instruktionen) möglich (jedoch keine «Current page»-Adressierung).

c) Indirekte Adressierung (deferred addressing)
Die adressierte *Speicherzelle enthält* die *Adresse* des Operanden, wobei die Adressierung der Speicherzelle mit den Methoden b, d erfolgen kann (IMP 8, IMP 16, MK 5065 P, PIP).

d) Indirekte Adressierung
Die Adresse des Operanden wird gebildet aus der *Summe* des Inhalts des *Indexregisters* (16 bit bei Computern mit 65 K-Adressierbereich) und dem in der Instruktion enthaltenen *Offset* (Displacement).
Diese Adressierungsart, verbunden mit Inkrementier- und Dekrementierfunktionen des Indexregisters ist sehr gut geeignet für Tabellenbearbeitungen (z. B. MC 6800, IMP 8, IMP 16, 2-Byte-Instruktionen).

e) Pointer-Adressierung (register indirect addressing)
Die Pointer-Adressierung ist eine Unterart der indexierten Adressierung ohne Offset, das heisst, die Adresse des Register(paares) entspricht der Operandenadresse. Diese Adressierungsart wird besonders bei den INTEL-Mikroprozessoren verwendet (z. B. H,L-Register bei MCS 8008 und MCS 8080).

f) Relative Adressierung
Die Adresse des Operanden wird aus der *Summe* des *Programmzählers* und dem in der Instruktion enthaltenen *Offset* (Displacement) gebildet (z. B. PIP, IMP 16, PACE).
Die meisten Mikroprozessoren erlauben die relative Adressierung *nicht* zur Adressierung von Operanden für Transferfunktionen, sondern zur Ermittlung der Sprungadresse (MC 6800, IMP 8, IMP 16) bei Sprungfunktionen (s. Kap. 12.2.3). Dabei liegt der Offset innerhalb eines Adressbereichs von ± 127 Bytes (2-Byte-Instruktionen). Während bei der relativen Adressierung von Operanden der Programmzähler nicht verändert werden darf, wird bei relativer Adressierung in Sprungfunktionen ein neuer Wert in den Programmzähler gesetzt.

g) Operand in der Instruktion (immediate addressing)
Der Operand ist in der Instruktion enthalten, das heisst, bei 8-bit-Mikroprozessoren enthält das 1. Byte den Operationscode und das 2. Byte den Operanden (MCS 8008, MCS 8080, MK 5065 P, MC 6800, PIP, IMP 8, IMP 16 usw.).

h) Chip-Selektierung, verschiedene Adressierung von RAM und ROM
Mikroprozessoren des Harvard-Typs (s. Kap. 2) unterscheiden zwischen Daten- (RAM) und Programmspeichern (ROM), entsprechend werden unterschiedliche Adressierungsmodes angewendet (z. B. MCS 4004, MCS 4040). Bei diesen von Tischrechnern abgeleiteten Mikroprozessoren ist die Adressierungsart ferner von der ursprünglich vorhandenen Gleitkomma-Zahlendarstellung beeinflusst. Beispielsweise muss beim MCS 4004 und MCS 4040 vor dem Zugriff von Daten im 80×4-bit-RAM vorerst mit einer Instruktion das RAM-Chip ausgewählt werden (select), ebenso wird mit der gleichen Instruktion ein mit Register bezeichneter Teilbereich (20×4 bit) des RAMs angewählt und innerhalb eines weiteren Unterbereiches des Registers (16×4 bit) eine 4-bit-Zelle adressiert. Erst in einer folgenden Instruktion wird festgelegt, ob in die adressierte Zelle oder in eines der restlichen 4 Statusworte (Speicherzellen 17...20 in einem Register) geschrieben oder daraus gelesen werden soll.

Zu den Transferfunktionen sind auch die Stackoperationen, welche den Inhalt von Registern und Flags auf oder vom Stack verschieben (Push, Pull) sowie die Ein-/Ausgabe-Operationen zu zählen.
Indexierte und relative Adressierung unterstützen adressunabhängige Programme beziehungsweise Programmierung (Position independent Code). Die Vorteile der indexierten und relativen Adressierung werden bei vielen Anwendungen stark eingeschränkt, falls die Offsets nicht unter Programmkontrolle ermittelt werden können. Diese Adressierungsverfahren verwenden meistens 2-Byte-Instruktionen (z. B. MC 6800), wobei das ebenfalls im ROM stehende 2. Byte den Offset enthält.
Bei programmabhängigen Verzweigungen via Adresstabellen (analog DO CASE *n*) oder analogen Tabellenmanipulationen mit *variablem Offset* ab Tabellenanfangsadresse sind die Offsets jedoch während «runtime» veränderlich, das heisst, der Inhalt eines Akkumulators sollte beispielsweise als Offset verwendet werden können.
Ferner ist es oft vorteilhaft, wenn Indexregister und Programmzähler ab Akkumulatoren geladen werden können (Ermittlung von Basisadresse durch eine Tabelle beziehungsweise Sprungadresse durch Algorithmus oder entsprechende Tabelle).

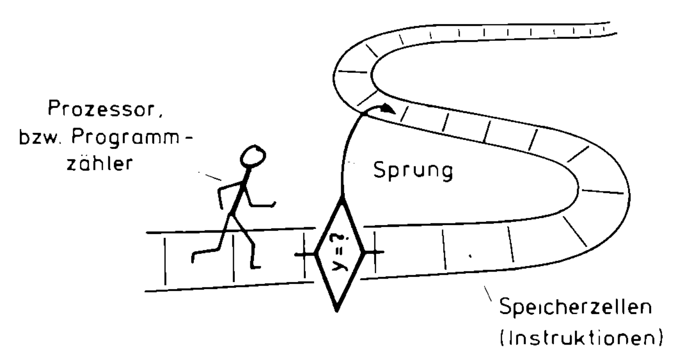


Bild 17 Veranschaulichung von Sprungfunktionen.
Bei den meisten Instruktionen wird der Programmzähler nach deren Ausführung inkrementiert, das heisst, die unmittelbar folgende Instruktion ausgeführt. Bei unkonditionellen Sprungfunktionen wird in jedem Fall eine neue Adresse in den Programmzähler geladen, während bei konditionellen Sprungfunktionen das Laden der neuen Adresse von einer bestimmten Bedingung (z. B. Resultat = Null) abhängig ist.

12.2.3 Sprungfunktionen

Nach der Ausführung von Instruktionen, welche nicht zu den Sprungfunktionen gehören, wird stets die unmittelbar folgende Instruktion adressiert (Ausnahme HALT-Instruktion). Hingegen erlauben Sprungfunktionen eine beliebige Veränderung des Programmzählers.

Es wird zwischen unbedingten und bedingten Sprüngen unterschieden. Bei bedingten Sprüngen wird eine bestimmte Kondition (z.B. Sign-, Zero-, Carry-, Parity-Flag, Bit n des Akkumulators, =, ≥, ≤, usw. getestet; je nachdem, ob die Sprungbedingung erfüllt ist oder nicht, wird an die neue Adresse gesprungen oder die folgende Instruktion ausgeführt. Die Effizienz des Instruktionssets ist sehr abhängig von den möglichen Sprungbedingungen. Zur Ermittlung der Sprungadresse sind

- absolute (fast alle Mikroprozessoren),
- relative (MC 6800, IMP 8, IMP 16, PIP),
- indirekte (IMP 8, IMP 16, MK 5065 P) und
- teilweise indexierte (IMP 16, IMP 8)

Adressierungsverfahren üblich. Damit die Begriffe mit Kapitel 12.2.2 übereinstimmen, muss hier die Sprungadresse als Operand betrachtet werden (und nicht deren Inhalt). Die Skip-Instruktionen (Überspringen der folgenden Instruktion) können als – speichersparender – Spezialfall der relativen Adressierungsart betrachtet werden.
Die relative Adressierungsart (Offset ± 127 Bytes) unterstützt das Schreiben von «relocatierbaren» Programmen (diese Programme können in beliebigen Speicherbereichen ablaufen).

Bei den Sprungfunktionen ist ferner zwischen JUMP- und CALL-(JSR)Instruktionen zu unterscheiden; bei CALL-Instruktionen wird nicht nur der Inhalt der Programmzähler verändert, sondern auch zusätzlich der alte Inhalt des Programmzählers auf den Stack gerettet. CALL-Instruktionen dienen zum Aufrufen von Subroutinen, die am Ende mit bedingten oder unbedingten RETURN-Instruktionen abgeschlossen werden. Die RETURN-Instruktionen verschieben den vorher auf den Stack geretteten Programmzählerinhalt wieder in den Programmzähler.
Sprunginstruktionen von 8-bit-Mikroprozessoren benötigen 2 (relative, indexierte Adressierung) bis 3 Bytes (absolute Adressierung).

12.2.4 Verschiedene Funktionen

Beispiele:

EI/DI	Enable/Disable-Interrupt
HLT	Halt des Computers
STC	Set carry

12.2.5 Kombination mehrerer Funktionen in einer Instruktion

Zur Erhöhung der Effizienz (Speicher-, Zeitbedarf) werden oft mehrere Funktionen in einer Instruktion vereinigt.
Zum Beispiel PPS 8:
- Addiere in Akku und Skip, falls Carry gesetzt wird.
- Lade Akku, inkrementiere Adresspointer X, vergleiche mit Adresspointer Y, Skip, wenn X = Y

IMP 8:
And-Operation, Skip, wenn Resultat = Ø

13. Elektrische Charakteristiken

Grundsätzlich wird annähernd allen Mikroprozessoren eine TTL-Kompatibilität zugeschrieben, doch gilt dies nur sehr beschränkt, z. B. «Fan Out» (1 bis 2 Low-Power-TTL-Loads), Temperaturbereich (0...70 °C) erfüllt nicht MIL-Spezifikationen, oft sind Pull-up-Widerstände notwendig usw.
Bei kleinen Mikroprozessorsystemen ist der Kostenanteil der Speisung bedeutend, deshalb ist es wünschbar, dass möglichst wenig Speisespannungen erforderlich sind. Beim MC 6800 wird für CPU, statische RAMs, PIA usw. nur eine 5-V-Spannung benötigt; bei Verwendung von billigen dynamischen RAMs, RePROMs, Keyboard-Encodern, Clock generator usw. werden trotzdem zusätzliche Spannungen erforderlich.

14. Bausteinsatz

Bei einigen Mikroprozessoren (z.B. Motorola, Rockwell, INTEL) wird ausser der Zentraleinheit ein ganzes Spektrum von vollkompatiblen Bausteinen angeboten (General purpose I/O, Asynchronous/synchronous Line Interface (UART), Modem, Keyboard-Encoder, Printer-Interface, Floppy-Disk-Controller, CRT-Controller, DMA-Controller usw.). Diese Bausteine reduzieren den Entwicklungsaufwand sehr stark und tragen zu einer weiteren Miniaturisierung des Systems bei.
Für einige Mikroprozessoren wäre es bei wenigen Zusatzelementen technisch möglich, auch Bausteine von anderen Prozessoren zu verwenden (z. B. ACIA, LSI-Modem von Motorola in INTEL-Mikroprozessor MCS 8080).

15. Technologie

Bei den erhältlichen Mikroprozessoren wird sowohl die erprobte PMOS-Technologie als auch NMOS-Technologie verwendet. Letztere wird noch nicht von allen Herstellern beherrscht, erlaubt jedoch eine um einen Faktor 2 bis 3 höhere Geschwindigkeit [6, 7].
Zurzeit sind ebenfalls 3 Mikroprozessoren in CMOS-Technologie angekündigt (COSMAC von RCA, IM 6100 von Intersil, AMC-8 von Asea-Hafo). Dies wird Vorteile hinsichtlich geringer Verlustleistung und variabler Speisespannung bringen.
Für schnelle Mikroprozessoren wird Schottky-TTL- (z.B. INTEL 3000) und neuerdings I^2L-Technologie (z.B. Texas SBP 0400) verwendet.

16. Zweitlieferant (Second Source)

Zurzeit ist für das Motorola-Mikroprozessorsortiment MC 6800 ein Zweitlieferant vorhanden.
Mindestens 4 Firmen haben angekündigt, dass sie den INTEL-Mikroprozessor MCS 8080 (in eigener Regie) nachbauen werden.
Da nicht alle Second-Source-Lieferanten eine Lizenz besitzen, ist die Kompatibilität vom Anwender sorgfältig zu prüfen.
Beispielsweise scheint der μ-COM 8 von Nippon zwar Software-kompatibel mit dem MCS 8080 zu sein, hingegen wird ein anderes Gehäuse (42 Pin) und eine andere Stift-Anordnung verwendet.

17. Software-Entwicklungshilfen
17.1 Allgemeines

Bild 18 stellt den Ablauf bei der Entwicklung eines Programms dar. Die vorhandenen Software-Entwicklungshilfen eines Mikroprozessorherstellers beeinflussen natürlich den Entwicklungsaufwand. Als grober Richtwert darf angenommen werden, dass 40% des Aufwandes für die Konzeption, 20% für Codierung (Editieren, Assemblieren) und 40% für Austesten und Dokumentation erforderlich sind.

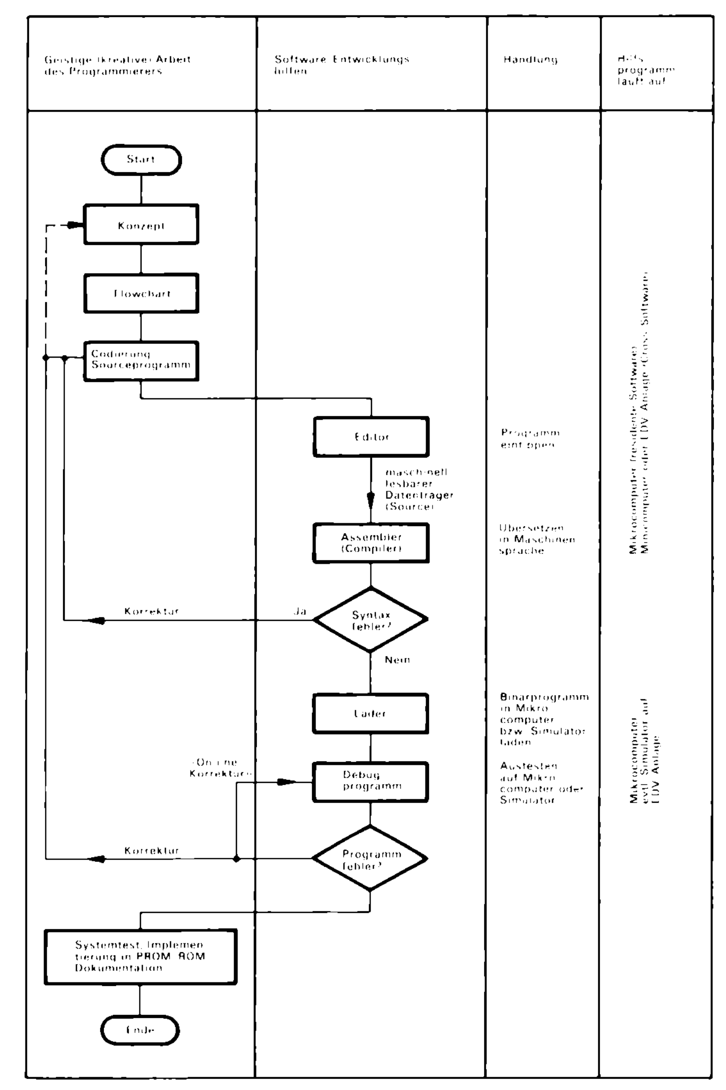


Bild 18 Veranschaulichung des Software Entwicklungsablaufs. Editor, Assembler, Lader und Debugprogramme sind minimale Voraussetzungen zur Softwareentwicklung.

17.2 Editor

Der Editor unterstützt das Eintippen und Korrigieren von Programmier- und Tippfehlern des Quellenprogramms (Sourceprogram). Grundsätzlich kann jeder Editor eines Mikro-, Minicomputers oder Rechenzentrums verwendet werden.

17.3 Assembler

Assembler übersetzen den Mnemocode (z.B. MOV B, A) der Instruktionen in den Maschinencode und setzen die richtigen Werte bei symbolischen Adressen und Operanden ein. Sie führen eine Syntaxanalyse durch (z. B. Fehlermeldung bei nicht vorhandener Instruktion oder fehlendem Register, zu grossem Zahlenwert, nicht definierten Adressen und Operanden usw.).
Assembler, welche auf dem Mikrocomputer selbst laufen, werden residente Assembler genannt, jene welche auf einem anderen System (Minicomputer, EDV-Anlage) arbeiten, werden mit Cross-Assembler bezeichnet.
Jeder Assembler hat die Möglichkeit, Mnemocodes in Maschinencodes zu übersetzen. Die Leistungsfähigkeit der Assembler der verschiedenen Computerhersteller unterscheidet sich vor allem hinsichtlich der vorhandenen Pseudoinstruktionen (Assembleranweisungen) und der Formatfreiheit bei der Eingabe.

Beispiele von Pseudoinstruktionen sind

ORG (Programmstart, Origin)
EQU (Gleichsetzen eines Symbols mit einem numerischen Wert, Equal)
END (Programmende)

und arithmetische und logische Operationen wie +, –, /, ×, MOD, AND, OR usw.
Die Pseudoinstruktionen bewirken lediglich eine Anweisung an den Assembler, sie erzeugen nicht ein der Anweisung entsprechendes Maschinenprogramm.
Wertvolle Pseudoinstruktionen bilden Makros und konditionelle Assemblierung. Im folgenden wird die Funktion dieser beiden Anweisungen beschrieben:

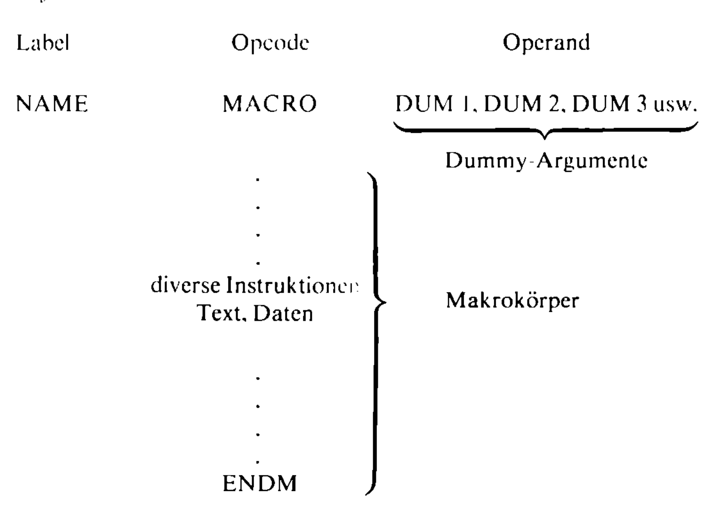
Bei der *konditionellen Assemblierung* werden die zwischen den Pseudoinstruktionen IF und ENDIF befindlichen Instruktionen, Daten und Texte vom Makroassembler ignoriert, falls der Ausdruck EXP Ø ist. EXP kann logisch oder arithmetisch verknüpft werden.

Aufbau:

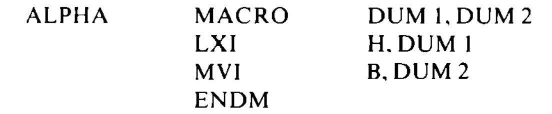
```
IF        EXP
.
.
.
diverse Instruktionen, Text, Daten
.
.
.
ENDIF
```

Bei *Makroanweisungen* können die zwischen den Pseudoinstrumenten MAKRO und ENDM befindlichen Instruktionen Daten oder Texte in einem Programm durch Aufruf des Makronamens eingesetzt werden, dabei werden die den Makroaufruf begleitenden realen Parameter in die im Makrokörper enthaltenen Argumente eingesetzt.

Aufbau:



Beispiel einer Makrodefinition:



Steht im Quellenprogramm irgendwo der Ausdruck

LABEL ALPHA 2,5

dann werden an dieser Stelle die im Makrokörper enthaltenen Instruktionen eingesetzt, wobei die Operanden reale Werte DUM 1 = 2 und DUM 2 = 5 erhalten.
Es können mehrere Makros ineinander verschachtelt werden, natürlich ist auch eine Kombination von Makros und konditioneller Assemblierung zulässig.
Ferner ist von Bedeutung, welche peripheren Geräte von Assemblern unterstützt werden.
Cross-Assembler auf Minicomputern mit Disk-Operating-Systemen bilden vor allem durch die leistungsfähige Peripherie sehr komfortable Software-Entwicklungsinstrumente.
EDV-Anlagen, welche im Batch-Betrieb arbeiten, sind infolge hoher «Turn around»-Zeiten (Zeit bis Resultate des Auftrags verfügbar sind) und nicht vorhandenen Lochstreifengeräten für die Cross-Assemblierung ungeeignet.
Timesharing-Anlagen von Rechenzentren ergeben bei grösseren Software-Entwicklungsprojekten sehr hohe Kosten (Rechenzeit, Input/Output-Belastung, Telefonspesen).
Die meisten Mikroprozessorenhersteller bieten in FORTRAN IV geschriebene Cross-Assemblerprogramme an. Diese Programme setzen meistens einen Computer mit einer Wortlänge von 32 bit voraus. Der Aufwand zur Modifikation dieser Programme auf einen 16-bit-Rechner (Minicomputer) ist schwer abschätzbar, meistens bildet ein Disk-Operating-System eine notwendige Voraussetzung. Bei der Implementierung auf hauseigene EDV-Anlagen ist die Verträglichkeit mit dem Betriebssystem zu beachten.
Es besteht die Möglichkeit, mit relativ geringem Aufwand vorhandene Assembler von Minicomputern für die Funktion eines Cross-Assemblers zu modifizieren [13], [15].
Bei dieser Methode entstehen Kompatibilitätsprobleme zu den Software-Hilfen (Assembler, Lader) des Mikroprozessorherstellers (unterschiedliche Pseudoinstruktionen, Format- und Zeichenvorschriften, Zahleneingabe, Binärformat des übersetzten Programms).

17.4 Compiler [14]

Ein Compiler ist ein Programm zum Übersetzen eines in einer höheren Programmiersprache geschriebenen Quellenprogramms in den Maschinencode.
Im Gegensatz zum Assembler, bei dem jeder mnemo-codierten Instruktion eine einzige Maschineninstruktion zugeordnet wird (ausser Makro), kann eine Instruktion in einer höheren Programmiersprache via Compiler eine grosse Anzahl von Maschineninstruktionen erzeugen.
Zurzeit ist nur von INTEL ein Compiler verfügbar, der Maschinencodes für die beiden 8-bit-Mikroprozessoren MCS 8008 und MCS 8080 erzeugt. Die höhere Programmiersprache wird mit PL/M bezeichnet und stellt ein Subset des bekannten PL/1 dar.
Der Compiler ist als Fortran-IV-Programm käuflich oder kann via Time-Sharing-Rechenzentrum von Honeywell-Bull verwendet werden.
Die Vorteile von PL/M gegenüber Assembler-Programmierung sind:
- Das gleiche Quellenprogramm kann Maschinencodes für den MCS 8008 und MCS 8080 erzeugen.

Es ist zu erwarten, dass diese Programmtransparenz auch bei künftigen Mikroprozessoren von INTEL verwendet werden kann.
Damit kann beim Einsatz eines neuen Mikroprozessors der grösste Teil der vorhandenen Anwenderprogramme unverändert übernommen werden.
- Selbstdokumentation:

Ein Quellenprogramm in PL/M ist relativ leicht lesbar, das heisst gut verständlich. Ein in Assemblersprache geschriebenes Programm hingegen benötigt viel Text und meistens auch Flussdiagramme, damit es lesbar beziehungsweise verständlich wird, was eine unbedingte Voraussetzung für spätere Behebung von Fehlern oder Modifikationen darstellt.
- Reduktion der Codierzeit:

Die gedrängte Schreibweise reduziert – für einen geübten Programmierer – den Aufwand zur Codierung des Quellenprogramms.

Nachteile sind:
- Die Implementation von PL/M auf dem Time-Sharing-Rechenzentrum von Honeywell-Bull ist für einen Mikroprozessoranwender nicht so leicht verständlich, das heisst, es müssen Kenntnisse des Rechenzentrumbetriebs erworben werden, welche mit der Mikroprozessoranwendung in keinem Zusammenhang stehen.
- PL/M erzeugt einen ineffizienten Maschinencode.

Von INTEL werden 10 bis 20% Zunahme des Speicherbedarfs angegeben (im Vergleich zur Assemblerprogrammierung). Kleine Programme mit vielen Bitmanipulationen ergeben jedoch viel höhere Ineffizienz. Eine Optimierung hinsichtlich Ausführungszeit (Run time) ist praktisch nicht möglich. Bei Programmen mit vielen Bitmanipulationen ist auch der Schreibaufwand des Quellenprogramms nicht wesentlich kürzer als bei der Assemblerprogrammierung.

- Das Debugging von PL/M-geschriebenen Programmen ist schwierig, da dies wiederum auf der Ebene der Assemblerinstruktionen (ohne Kommentar) bewerkstelligt wird und somit das vom Compiler übersetzte Programm vom Anwender verstanden werden muss.
- Interrupt-aktive und zeitkritische Aufgaben sind mit PL/M schwierig lösbar, da der Programmierer die Übersicht über die zeitlichen Vorgänge des Prozessors verliert.

Die Verwendung von höheren Programmiersprachen im Bereich von Realtime-Aufgaben (Prozessrechner) ist ein alter Traum der Systemprogrammierer. Obwohl schon viele Versuche gemacht wurden, ist bis jetzt noch keine allseits befriedigende Lösung gefunden worden.

Auch andere Mikroprozessorenhersteller werden künftig Compiler anbieten (National Semiconductor auf IMP 16), ferner wird u.a. an skandinavischen Universitäten an diesem Problem gearbeitet.

Bei der Bewertung der Nützlichkeit einer höheren Programmiersprache sind die Eigenschaften von

- Sprache
- Compiler (Optimierungsstufen)
- Implementierung des Compilers im Time Sharing-Rechenzentrum

ausschlaggebend.

Beim üblichen Anwendungsbereich von Mikroprozessoren ist die Anwendung von Compilern der Verwendung von Makroassemblern – zusammen mit strukturierter Programmierung – gegenüberzustellen.

17.5 Lader

Ein Lader ist ein Hilfsprogramm zum Laden von übersetzten Programmen (Binärprogramme) ab maschinell lesbaren Datenträgern beziehungsweise des entsprechenden Lesegerätes (z.B. Lochstreifenleser) in den Arbeitsspeicher. Der Lader ermittelt die Adressen des zu ladenden Programmes, konvertiert den auf dem Datenträger codierten Maschinencode, prüft die Quersummen, startet das geladene Programm usw.

Beim IMP 16 von National Semiconductor ist ein relocatierfähiger Lader vorhanden. Dieser erlaubt das Zuweisen der absoluten Adressen des Programms während der Ladezeit. (Bei den «gewöhnlichen» Ladern muss diese Zuweisung während der Assemblerzeit erfolgen.)

17.6 Debugprogramm

Debugprogramme unterstützen das Austesten von Anwendungsprogrammen auf dem Computer.

Sie erlauben, den Computer an vorgegebenen Adressen anzuhalten, den Inhalt von Registern, Flags und Speicherzellen octal oder hexadezimal auf der Konsoleschreibmaschine auszugeben und von dort aus zu ändern (via Tastatur).

Ferner kann der Speicherinhalt im Binärformat ausgegeben werden.

Die Eigenschaften der Debugprogramme verschiedener Mikroprozessorenhersteller unterscheiden sich vor allem hinsichtlich der Anzahl einsetzbarer Haltepunkte (Breakpoints) sowie allfälliger Loop-Zähler (Passcount) zum Anhalten nach einer vorgegebenen Anzahl durchlaufener Schlaufen.

17.7 Simulatorprogramm

Ein Simulatorprogramm erlaubt die Simulation eines Mikroprozessors auf einer EDV-Anlage und dient zum Austesten von Anwenderprogrammen. Entsprechend dieser Aufgabe sind die gleichen Grundfunktionen vorhanden wie bei einem Debugprogramm, es sind jedoch einige zusätzliche Möglichkeiten implementiert (z.B. Anhalten, falls der Inhalt einer vorgegebenen Speicherzelle ändert, Ausgabe der Ausführungszeit [Run time] usw.).

Hingegen müssen die Eingabe von aussen (Input) und Interrupts simuliert beziehungsweise vorprogrammiert werden.

Der Simulator von Motorola (für MC 6800) simuliert auch die I/O-Bausteine (z.B. PIA).

Da bei vielen Entwicklungen – zusammen mit der Software – auch Hardware-Moduln ausgetestet werden, bietet die Anwendung von Simulatorprogrammen selten Vorteile.

18. Tabellarische Übersicht der erhältlichen und angekündigten Mikrocomputer

Die folgende Zusammenstellung der wichtigsten Parameter erlaubt einen groben Vergleich der verschiedenen Mikrocomputertypen.

Es ist nicht einfach, ganz unterschiedliche Strukturen in ein gemeinsames Schema zu pressen, oft sind bei einem Parameter verschiedene Interpretationen möglich. Bei vielen angekündigten Mikrocomputern sind die Unterlagen noch unvollständig. Die Angaben hinsichtlich Bausteinsortiment und Software sind dauernden Änderungen unterworfen. Der Zeitpunkt der Verfügbarkeit kann sich aufgrund bisheriger Erfahrungen um Jahre verzögern.

Aufgrund der Daten der lieferbaren und angekündigten Mikrocomputer ist der folgende Trend feststellbar:

- 8-bit-Operandenwortlänge
- 65 K-Bytes-Adressbereich
- 1-, 2-, (3)-Byte-Instruktionswortlänge
- separater Adress- und Datenbus
- N-MOS-Technologie
- mehrere Adressiermodes zur Adressierung von Operanden und für Sprunginstruktionen
- DMA-Möglichkeit
- Standard RAM-, PROM- und ROM-Elemente für Speicher
- Bausteinsatz für I/O-Interfaces, Taktgenerator usw.
- steigende Softwareunterstützung (Makroassembler, Debugprogramme, Software-User Club, PL/M-ähnliche High-Level-Language).

Die Übersicht wurde aufgrund von Unterlagen der Mikroprozessorenhersteller zusammengestellt.

Ergänzende Literatur findet sich u. a. in [8, 9, 10, 11, 12], siehe Literaturverzeichnis auf Seite 55.

Tabellarische Übersicht der erhältlichen und angekündigte

Kennwerte — 18.1 4-bit-Mikroprozessoren

Hersteller		INTEL **1**	INTEL **1**	Rockwell **1**	National Semicond. **2**	Nippon **14**	Texas
Typ		MCS 4004	MCS 4040 **3**	PPS 4	IMP 4	μCOM 4	TMS-10
Verfügbarkeit		1971	4. Q. 1974	1973	4. Q. 1974		
*E*in-/*M*ultichip CPU		E	E	E	M. 3 Chip **4**	E	E
Wortlänge in bit	Daten	4	4	4	4	4	4
	Instruktionen	8 (16)	8 (16)	8 (16)	8, 12, 16, 20	8 (16)	8
Adressierbereich	Daten	1,28 K	1,28 K	4 K (8 K) **5**	4 K × 4 bit **9**	4 K × 4 bit	256 × 4
	Instruktionen	4 K	8 K	4 K (16, 32 K)		8 K × 8 bit	1K × 8
Anzahl Instr. nach Herstellerangaben		46	60	50	42	55	43
Anzahl freiprogr. Register (ohne Stack/PC)		16 × 4 bit + Accu	24 × 4 bit + Accu	Adr. point 3 × 4 X-Reg. 4 bit + Accu	4 × 4 bit	12-bit-Adr.-point. + Accu	6-bit-Ac point. -
Anzahl adressierbare IN/OUT-Devices		16 × 4 bit I/O **8** 16 × 4 bit OUT	32 × 4 bit I/O **8** 16 × 4 bit OUT	49 × 4 bit OUT 50 × 4 bit IN	4 K × 4 bit **9**	240 × 4 bit **15**	IN: 4 bit OUT: 9 bit via PLA
Stack	*I*ntern/*E*xtern Grosse	I 3 × 12 bit	I 7 × 12 bit **6**	I (E) 2 × 12 bit **7**	I 16 × 4 6 × 12 bit **18**	E 32 × 4 bit	I 1 × 12 bit
*s*ynchron/*a*synchron		s	s	s	s	s	s
Interrupt		Nein	1 Level	Nein	1 Level	Nein	Nein
Anzahl Stifte der CPU		16	24	42	24, 40, 24	28	28 oder
Technologie		P MOS	P MOS	P MOS	P MOS		P MOS
Speisespannung CPU in V		· 5, 10	· 5, 10	· 5, 12	· 5, 12	+12, +5, 5	15
Standard RAM- oder ROM-Elemente anwendbar		**10**	**10**	**10**	Ja	Ja	Nein
Anzahl TTL Chips für Mininalsystem		keine	keine	keine	wenige	ca. 10	keinen auf Chip
Ausführungszeit/Instr. in μs		10,8 21,6	10,8 21,6 **11**	4 12	ca. 10 26	ca. 5 10	12
Adressiermodes Operanden	direct				×		
	abbreviated					× (Stack)	
	indirect						
	pointer	×	×	×	×	×	×
	index				×		
	relativ						
	immediate	×	×	×	×	×	
Adressiermodes Sprung instr.	direct	×	×		×		
	abbreviated			×			×
	indirect			× (limited)			
	index					×	
	*r*elativ/*s*kip	s	s	s		s	
Testbare Flags	zero, carry, sign	Z, C	Z, C	Z, C	Z	Z, C	C, Z, S **11**
	overflow	× Testline	× Testline			× Testline	
	parity						
				×	×		
					×		
	n^{tes} Bit				×		
	progr. Flag			2 **12**		1 **12**	
Second Source		National Semicond.					
Software	Resident. *Ass./Ed./Deb*	A, D	A, D	A, D	D		
	Cross Assembler	×	×	×	×		×
	Compiler						
	Simulator progr.	×	×				
Bausteinsortiment (exkl. Speicher)		gut	sehr gut	sehr gut			
Bemerkungen							Hardware simulator vorhanden

(Die halbfett gedruckten Zahlen geben Hinweise auf die ergänzenden Bemerkungen in den entsprechenden Kolonnen)

1 Harvard Typ (s. Kap. 2)
2 Von Neumann Typ (s. Kap. 2)
3 Aufwärtskompatibel zu MCS 4004
4 Registereinheit RALU, Mikroprogrammeinheit CROM; Erweiterung von Programmzähler, Implementierung der beim GPC/P System notwendigen externen Logik (FILU)
5 Bei Adressexpansion mit diskretem Outputport der CPU
6 CPU Status kann bei Interrupt in 8 × 4 bit Registerbank gerettet werden
7 Interner Stack kann bei Überlauf extern abgespeichert werden
8 Bei I/O über RAM- und ROM-I/O-Ports und maximaler RAM/ROM Bestückung
9 Speicher- und I/O-Adressen zusammen auf ... beschränkt
10 Bei Verwendung eines speziellen «Standard Mem... Interface Chips» (4289 bei INTEL) können Standardsp... cherelemente verwendet werden, sonst stehen Spez... RAM und ROM zur Verfügung
11 Schnelle Version: 6,7 13,2 μs
12 Flags können unter Progr. Kontrolle gesetzt werden
14 Angaben noch unvollständig
15 Anzahl I/O Ports hängt von der RAM Bestückung ab
16 RAM und ROM auf CPU Chip
17 Setzen gemeinsames Statusbit
18 Stack für Register 16 × 4 bit
Stack für Programmzähler 6 × 12 bit

Mikroprozessoren, September 1975

(Die halbfett gedruckten Zahlen geben Hinweise auf die ergänzenden Bemerkungen in den entsprechenden Kolonnen)

18.2 8-bit-Mikroprozessoren

INTEL	INTEL	Motorola	Rockwell	National Semicond.	National Semicond. **15**	Mostek	Signetics **4**	RCA	ASEA HAFO **4**	AEG-Tel. + Gen. Instr. **4**
MCS 8008	MCS 8080	MC 6800	PPS 8	IMP 8	CMP 8	MK 5065 P	PIP 2650	COSMAC	AMC-8	CP3-F
1. Q. 1972	1. Q. 1974	4. Q. 1974	4. Q. 1974	2. Q. 1974		2. Q. 1975		3. Q. 1975		1. Q. 1975
E	E	E	E	M, 3 Chip	E	E	E	2 Chip **3**	E	E
8	8 (16) **8**	8 (16) **8**	8	8	8	8	8	8 (16) **15**	8	8
8 (16, 24)	8 (16, 24)	8 (16, 24)	8 (16, 24)	8 (16)	8 (16)	8 (16)	8 (16, 24)	8 (16)	8 (16)	8 (16)
16 K	65 K	65 K **3**	16 K **5**	65 K	65 K	32 K	32 K	65 K	1/4 K 1 K	16 K
48	78	72	90	38 **17**	ca. 50	51	75	59	42	48
7 × 8 bit	7 × 8 bit	Index r. 16 bit 2 Accu × 8 bit	Accu, X, Y, Z je × 8 bit	4 × 8 bit	5 × 8 bit 2 × 16 bit	3 Accu	2 banks zu 3 Register + Accu	16 × 16 bit + 3 × 4 bit	5 × 8 bit ?	48 × 8 bit + 4 Reg. × 8 bit **5**
24 OUT 8 IN	256 OUT 256 IN	65 K **3**	15 parallel I/O LSIs zu 2 Byte **4**	65 K	256 OUT 256 IN **13**	16 OUT 16 IN	256 OUT 256 IN	8 OUT 8 IN	64 I/O	64 I/O ?
I 7 × 14 bit	E 65 KBytes	E 65 KBytes	I (1 Level) + E 32 Bytes	I 16 Bytes	E 65 K	E 256 Bytes	I 8 × 15 bit **10**	I (Reg. array) E (erweiterb.)	I 3 × 10 bit	I 3 × 14 bit
a	a	s	s	s	a	s	a	s	a	
1 **16**	1 **16**	2 + SWI **2**	3 resp. 17 **6**	1	2	2	Ja, Vektor	1 **16**	1	
18	40	40	42	3 × 24	40	40	40	28 + 40	40	40
P-MOS	N-MOS	N-MOS	P-MOS	P-MOS	N-MOS	P-MOS	P-MOS	C-MOS	C-MOS	?-MOS
+5, 9	+5, 5, 12	+5	+5, 12	+5, 12	+5, 5, +9	+5, 5, 12	+5	5 12	5 15	5, 12
Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
ca. 20	wenige	nur Taktgener.	keine	ca. 10 20	wenige	wenige **12**	wenige	einige CMOS-Elemente **12**	wenige	wenige
20-44 **9**	2 8,5	2 12	4 12	4 21	typ. 2, 4	4 30	≈5 10	8	1,6 3,2	5 20
	•	•	• **11**				•			
		• (page Ø)	•	•		•				
				•	•	•	•			
•	•		•					•	•	•
		•		• **12**	•					
							•			
•	•	•	•	•	•	•	•	•	•	•
•	•	• (non cond.)	• **11**			innerhalb 4 K	•	• **18**	• ?	•
			•	•	• ?	• **2**		• **17**		
	• üb. H,L-Reg.			•	•	**1**	•			
		• (non cond.)		•			•			
		r (cond.)	s	s (cond.)	r, s	s (cond.)	r			
Z, C, S	Z, C, S	Z, C, S	Z, C, S	Z, C, S	Z, ?	Z, C, S	Z, C	Z, C	C, ?	Z, C, S
		•		•		• 1 Testline	•	• 4 Testlines	• 4 Testlines	
•	•									
		•	•	•		•	•		•	
		•					•			
			•	•	• ?					
				Testline						
MIL **1**	MIL, AMD MO, NI, TX **1**	AMI **7**								
A, E, D	A, E, D	A, E, D **10**	A	A, E, D	D	A, E, D		A, E, D		
•	•	•	•	•	•	•	•	•	• ?	•
PL/M	PL/M									
•	•	•	•			•		•		• •
	gut	gut	sehr gut							
					data-comm. orientiert	Triple level structure		DMA-Addr. Register on CPU **11**		

1 MIL Microsystems International Canada (Betrieb im 1. Q. 1975 eingestellt!) AMD American Micro Devices, MO Mitsubishi Electric Corp. and Oki Electric Ind., NI Nippon Electric Company u. COM 8, TX Texas Instruments TMS 8080
2 Bei einem Interrupt wird der gesamte CPU Status (Akkus, Flags, Indexreg., Progr. zähler) hardwaregesteuert in den Stack transferiert. SWI Softwareinterrupt
3 I/O- und Speichertransfer wird nicht unterschieden
4 Aufgrund des Instruktionsformates sind 127 Out- und 127 Input Devices adressierbar. LSIs für 4- und 8-bit Parallel Datentransfer vorhanden. LSIs enthalten Interrupt Logik, Status- und Controlregister
5 ROM- und RAM Zugriff sind zeitlich überlappend
6 Mit Level 3 können alle 15 GP I/O-Bausteine, auf der Basis einer Priority Chain (s. Kap. 10.2) verbunden werden
7 American Microsystems International

8 Doppelregister (MCS 8080) bzw. Indexregisteroperationen weisen 16 bit Operanden auf
9 Schnelle Version MCS 8008 1 12,5 27,5 µs
10 Bezeichnung für Debuggingprogr.: EXBUG
11 Oft verwendete Sprungadressen und Operanden können in gemeinsamen Data Command- und Literal-Pool abgespeichert werden. Dies reduziert den Speicherbedarf
12 Indexregister AC 3 8 bit! (Most sign. Byte)
13 Transfer zwischen Speicherzelle und I/O Bus mit 1 Instruktion (ohne Zwischenspeichern in Accu)
15 Angaben noch unvollständig
16 Extern auf 8 prioritätsverknüpfte vektorielle Interrupts erweiterbar
17 Instruktionsset durch zusätzliche CROMs erweiterbar

1 Unconditional 2 Byte Instruktion: indirekte Adressierung via current page oder page Ø (Offset im 2. Byte der Instr.). 1 Byte Instruktion: Offset für indirekte Adressierung in current page oder Ø ist in Akku enthalten (page 256 Bytes)
2 Unconditional 1 Byte Instruktion: Sprungadresse (Low Byte) ist in Akku enthalten. (Current page oder page Ø Adressierung)
3 Später 1 Chip-Version: Zur Zeit enthält 1 Chip (40 pin) die CPU und ein zweiter Chip (28 pin) das Register array
4 Nur provisorische, unvollständige Daten vorhanden
5 48 × 8 bit scratchpad memory («Notizblock»)
6 Programmzähler und RAM Adresspointer befinden sich auf ROM und RAM Chip. Mit Standardmemoryinterface-Chip sind Standard-Speicher verwendbar.
7 Interrupt auf ROM Chip, Programmzähler (PC), Adresspointer, Timer, 1 Level PC Stack und 2 I/O-Ports auf ROM-Chip

9 Taktgenerator auf CPU-Chip
10 Stack nur für Programmzähler
11 Arbeitet im Temperaturbereich 55 C
12 Adressregister und Taktgenerator m gebaut werden
13 64 × 8 bit scratchpad memory
14 Unter anderem ist ein Speicherinterf mische RAM und ein DMA-Controller Ch
15 16 bit Inkrement und Dekrement Op
16 Simultane Bearbeitung von mehrere möglich
17 Immediate Operand ersetzt Low Byte ler
18 Direkte Adressierung innerhalb 6 Umschaltung auf einen anderen Program
19 Fortran

18.3 12-bit-Mikroprozessoren

18.4 16-bit-Mikroprozessoren

Fairchild			Toshiba **3**	Intersil **3**		National Semicond.	National Semicond.	General Instr. **12**	Nippon **12**	
F 8			TLCS 12	IM 6100 **5**		IMP 16 **4**	PACE **4**	CP 1600 **9**	μ COM 16	
2. Q. 1975			4. Q. 1974			3. Q. 1973	2. Q. 1975			
E **6**			E	E		M/5 Chip **3**	E	E		
8			12	12		16 (8)	16 (8)	16	16	
8 (16, 24)			12	12		16	16	16 **11**		
65 K			4 K **2**	4 K 32 K		65 K **10**	65 K **10**	65 K **10**	65 K	
70			80 (108)	· 40		· 43 **3**	45	68		
64 × 8 bit + Accu **13**			6 × 12 bit	2 × 12 bit		4	4	8		
256 I/O			4 K **2**	64		65 K	65 K **4**	65 K **4**		
I 1 Level + ext. erweiterb.			E **10**	**5**		I 16 × 16 bit	I (E) 10 × 16 bit **5**	E 65 K		
s			a	a		s	a			
1 Level/ROM **7**			8 maskierbar	1		1	4 (mask.)			
40			42/36 **11**	40		5 × 24	40	40		
N-MOS			P MOS	C MOS		P-MOS	P-MOS	N-MOS		
+5, -12			5, +5	3 7		+5, 12	+5, -12			
6			Ja **9**	Ja		Ja	Ja			
keine **9**			wenige			ca. 16	**6**			
2 13			10 20 **8**	5 11		4,2 14	typ. 10			
			·							
				·						
			·	·		·	·			
·										
			·			·	·			
			·			·	·			
·			·			· (8 bit)	× (8 bit)			
·			· ?							
				·						
				·		·	·			
						·	·			
r			r ?	s		r, s	r, s			
Z, C, S			Z, C, S	Z, C, S		Z, C, S	Z, C, S			
·						·	·			
			·			+ 4 Testlines	+ 3 Testlines			
Mostek										
A, D			A, E	A, E, D **6**		A, E, D **2**	A, E, D **2**			
·			·			·	·			
· **19**						PL/N **7**	PL/N **7**			
·			·							
14										
			4	**7**		**8**				

Cbis + 125°C
müssen diskret auf-

face-Chip für dyna-
hip vorhanden
perationen
en Interrupts nicht

e in Programmzah-

55 KBytes durch
mzähler

2 Kein Unterschied in Speicher und I/O Adressierung
3 Unterlagen preliminar
4 Temperaturbereich von 20 C bis + 80 C. Multiply/Divide Instruktionen im Standard Instr. set
5 PDP8 kompatibel, deshalb mussen Autoindexregister (Adr. 10 17) und Subroutinen-Anfangsadressen in RAM (nicht in ROM) stehen! (s. Kap. 9)
6 Es kann die gesamte PDP8 Software von DEC verwendet werden
7 Arbeits Temperaturbereich 55 C bis + 125 C
8 Ausser Multiply/Divideinstruktionen, welche 40 bis 60 μs benotigen.
9 Memory Control Unit erforderlich
10 Bei jeder Subroutine steht am Anfang eine Adresse, die auf eine RAM Speicherzelle hinweist. Bei einem Sprung in die Subroutine wird der laufende Programmzahler in dieser Speicherzelle zwischengespeichert
Die RAM Speicherzellen mit Adresse 8 bis 15 enthalten die Startadressen (Vektoren) der den Interrupteingangen (0 bis 7 zugeordneten Interruptserviceroutinen. Bei einem Interrupt wird der Inhalt des laufenden Programmzahlers mit dem entsprechenden Vektor ausgetauscht (damit sind keine «reentrant» Programme moglich).
11 Neuer Chip mit 36 Pin

2 + relocatierbarer Lader
3 Bei zusatzlichen CROM fur die Erweiterung des Instruktionssets vergrossert sich die Chip Zahl. Die von National Semiconductor erhaltlichen Zusatz-CROM unterstützen vektorielle interrupts (s. Kap. 3.2), Doppelprazisionsarithmetik, Blocktransfer, Memory search, mathematische Routinen, usw.
4 I/O und Speicheradressierung ist identisch
5 Stackinhalt kann in Arbeitsspeicher verschoben werden
6 Adressregister und Taktgenerator
7 residenter Compiler!
8 Alle IMP Prozessoren arbeiten im Temperaturbereich 50 C bis + 85 C
9 Architektur des CP 1600 wurde von Honeywell entwickelt
10 Adress und Datenbus gemeinsam
11 8 bit Struktur intern
12 Angaben unvollstandig

18.5 Übersicht «Single board processors» (Naked Minis)

Hersteller	Typ	Wortlänge in bit	Kompatibel zu	Bemerkungen
Digital Equipment	PDP 8/A	12	PDP 8	13. bit für RAM-Adressierung bei JSR-Autoinkrementinstr. usw.
Digital Equipment	LSI 11	16	PDP 11/40	LSI-Hersteller: Western Digital. CPU auf 4 × 40 pin MOS-LSI
General-Automation	LSI 12 LSI 16	12/8 16	SPC 12 SPC 16	Entwicklung dieses Computers vorläufig eingestellt, da Rockwell die SOS-Chips für die CPU nicht liefern konnte
Computer Automation	Naked mini LSI	16	α-16	2 Varianten: LSI 1: 7 Chip-P-MOS-CPU, ähnlich zu GPC/P-Mikrocomputer von NSC LSI 2: CPU besteht aus Standard LSIs
National Semiconductor	IMP 16	16	–	s. Kapitel 3.4
Data General	NOVA 2/4	16	NOVA-Serie	
Scientific Micro Systems	Micro-controller	16		Schottky TTL-LSIs, nur 8 Instruktionen

18.6 Übersicht mikroprogrammierbare Mikroprozessoren

Hersteller	Typ	Wortlänge des Register-chips in bit (Slice)	Möglicher Wort-längen-bereich d. Computers in bit	Vorhandene Elemente:				Techno-logie	Mikroinstr. zyklus in ns	Unter-stützung durch Hersteller	Bemerkungen
				Mikroprogr.-controlunit	Mikroprogr.-speicher	Register-einheit ALU	andere				
National Semicond.	GPC/P	4	4 32	CROM (100 × 23 bit)		RALU		P-MOS	1400	Mikro-progr.-assembler, FACE (erlaubt Simulat. d. CROM)	IMP 4, IMP 8, IMP 16 wurden mit GPC/P realisiert
INTEL	3000	2	max. 320 (Jedes Vielfache von 2 bit)	MCU 3001	Standard PROM oder ROM, μ-Instr.-Wortlänge variabel	CPE 3002	Lock ahead carry gen. 3003 Interrupt Contr. 3214	Schottky bipolar	ca. 100–120	Mikro-progr.-assembler	Zweitlieferant: Signetics
Monolithic Memories	5701/6701	4	Jedes Vielfache von 4 bit 4...64	muss diskret aufgebaut werden (s. Bilder 3, 4, 5)	Standard PROM oder ROM, μ-Instr.-Wortlänge 24 bit	Mikro-contr. 5701/6701	Look ahead carry gen. 54 182	Schottky bipolar	200	?	36 μ-Instr.
Texas Instr.	SBP 0400	4	Jedes Vielfache von 4 bit	muss diskret aufgebaut werden (s. Bilder 3, 4, 5)	Standard PROM oder ROM, μ-Instr.-Wortlänge 9 bit	Binary Prozessor Element SBP 0400	Look ahead carry gen.	I^2L	110–530	?	
Western Digital	MCP 1600	8	8, 16	CP 1621 B Control Chip	CP 1631 B Microm 512 × 22 bit max. 4 Chips	CP 1611 B Data Chip	–	N-MOS	300–600	Mikro-progr. assembler und Simulator	LSI 11 von DEC wurde auf der Basis von MCP 1600 realisiert
Advanced Micro Devices	AM 2901	4	Jedes Vielfache von 4 bit	muss diskret aufgebaut werden	Standard PROM oder ROM, μ-Instr.-Wortlänge 9 bit	AM 2901		Schottky bipolar	100	?	
Motorola	MC 10800	4		–	–	–		ECL	55		Angaben noch unvollständig

Der Algorithmus

Der Begriff Algorithmus wird anhand eines Beispiels erläutert. Neben der normierten Flussdiagrammdarstellung wird am Beispiel auch eine vereinfachte Flussdiagrammsymbolik vorgeschlagen.

Immer mehr Fachkräfte sind damit beschäftigt, Vorgänge – auch Prozesse genannt – zu analysieren und so in Ablaufschemata zu fassen, dass sie einem Automaten zur Steuerung oder Durchführung übergeben werden können.
Ein immer wiederkehrendes Mittel, einen Vorgang «computergerecht» zu beschreiben, ist der Algorithmus und seine Darstellung durch ein Flussdiagramm, das hier einmal in der DIN-Symbolik und als Vorschlag auch in einer vereinfachten Symbolik gegeben ist.
Obwohl es bisher nicht gelungen ist, exakt zu definieren, was im allgemeinsten Sinne unter einem Algorithmus zu verstehen ist, kann man doch angeben, welche Eigenschaften er aufweist:

Beginn
Bereitstellen der Eingangsgrössen *E*
Durchführen der Regeln und Rechenvorschriften
neue *E* aus *A'*
vorläufige Ausgangsgrössen *A'*
Algorithmus aufgrund einer Entscheidungsregel beendet?
Nein
Ja
definitive Ausgangsgrössen *A* aus *A'*
Stop

Bild 1 Allgemeines Ablaufdiagramm eines Algorithmus

Ein Algorithmus liegt dann vor, wenn eine Anzahl Eingangsgrössen durch ein System von Regeln und Rechenvorschriften so behandelt wird, dass eine Anzahl gewollter Ausgangsgrössen entsteht. Hierbei ist die in Bild 1 gegebene, grobe Ablaufstruktur zu beachten.
Wie man aus Bild 1 erkennt, ist das System der Regeln und Rechenvorschriften der massgebende Teil für die Durchführung eines Algorithmus. Anhand eines klassischen mathematischen Beispiels soll dies verdeutlicht werden.
Der sogenannte «Euklidische Algorithmus» ermöglicht es, den grössten gemeinschaftlichen Teiler (ggT) zweier Zahlen herauszufinden, ohne die beiden Zahlen in Primfaktoren zerlegen zu müssen.
Gegeben seien die beiden Zahlen a und b, wobei $a \geqslant b$. Gesucht ist der ggT von a und b.

Algorithmus (Rechenverfahren):
$a : b = Q_1$, Rest R_1
$b : R_1 = Q_2$, Rest R_2
$R_1 : R_2 = Q_3$, Rest R_3
$R_2 : R_3 = Q_4$, Rest R_4 usw.
Der Divisor, bei dem die Division aufgeht (das heisst Rest $R = 0$), ist der ggT.

Zahlenbeispiel: $a = 3080$, $b = 2345$ ($a > b$)
3080 : 2345 = 1, Rest 735
2345 : 735 = 3, Rest 140
735 : 140 = 5, Rest 35
140 : **35** = 4, Rest 0; die Division ist aufgegangen.
Der ggT von 3080 und 2345 ist 35.
Um diesen Rechenablauf nun übersichtlich darzustellen, verwenden wir die graphische Symbolik des Flussdiagramms, wie sie in der Datenverarbeitung üblich ist. Die wenigen Symbole, die man benötigt, um ein Flussdiagramm aufzustellen, sind in Bild 2 zusammengefasst:

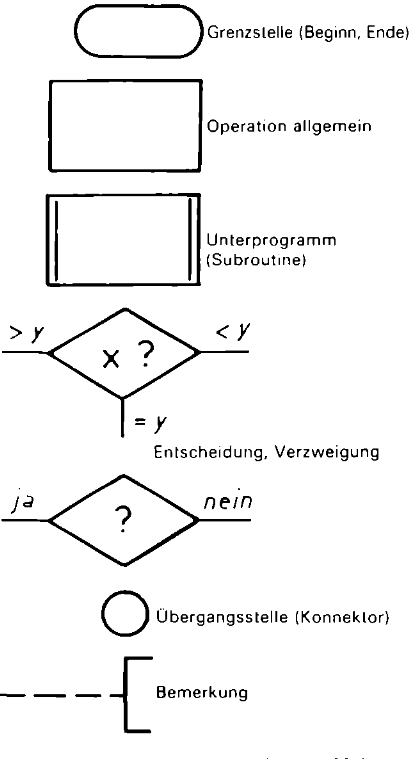


Bild 2 Flussdiagrammsymbole (DIN 66001)

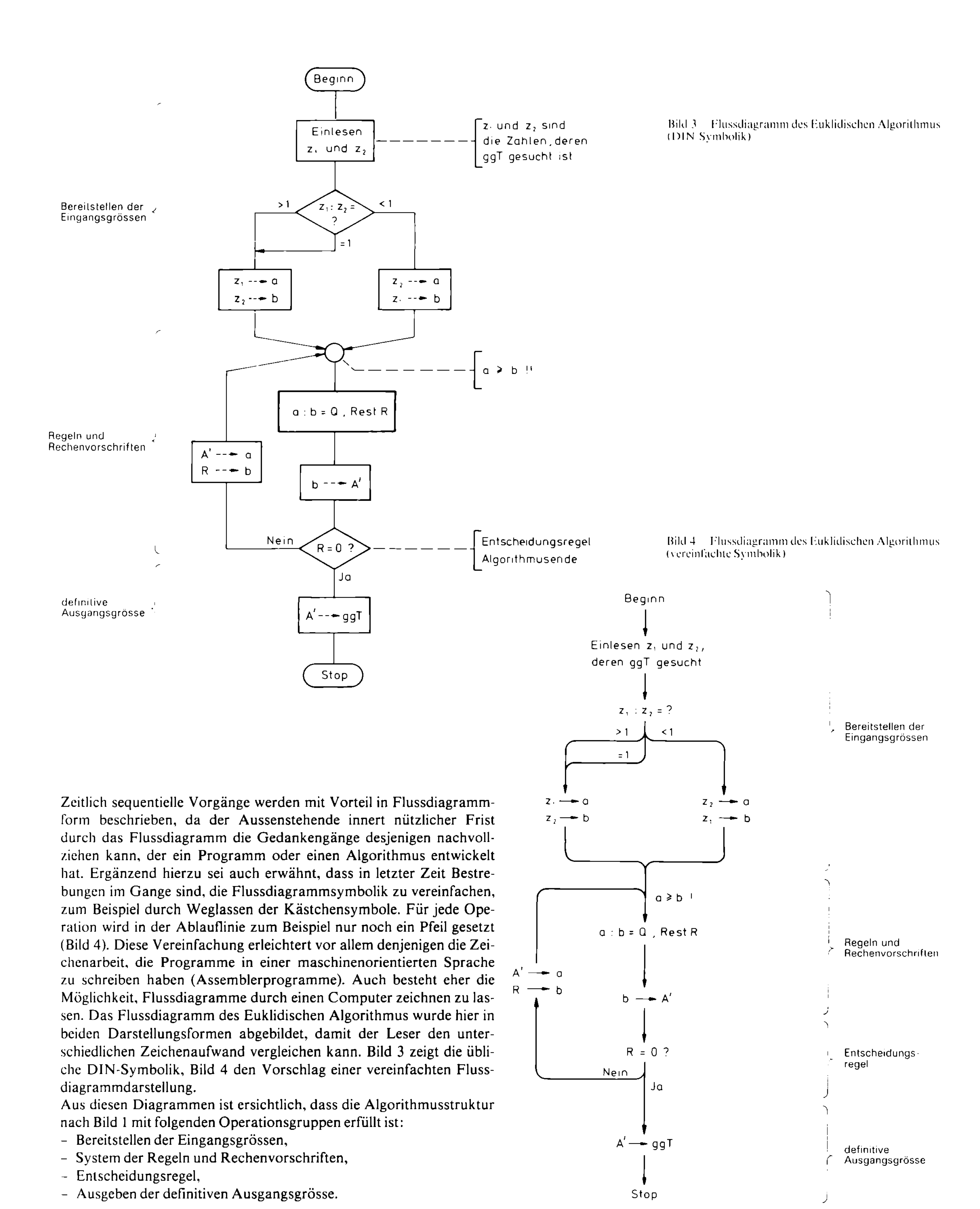


Bild 3 Flussdiagramm des Euklidischen Algorithmus (DIN Symbolik)

Bild 4 Flussdiagramm des Euklidischen Algorithmus (vereinfachte Symbolik)

Zeitlich sequentielle Vorgänge werden mit Vorteil in Flussdiagrammform beschrieben, da der Aussenstehende innert nützlicher Frist durch das Flussdiagramm die Gedankengänge desjenigen nachvollziehen kann, der ein Programm oder einen Algorithmus entwickelt hat. Ergänzend hierzu sei auch erwähnt, dass in letzter Zeit Bestrebungen im Gange sind, die Flussdiagrammsymbolik zu vereinfachen, zum Beispiel durch Weglassen der Kästchensymbole. Für jede Operation wird in der Ablauflinie zum Beispiel nur noch ein Pfeil gesetzt (Bild 4). Diese Vereinfachung erleichtert vor allem denjenigen die Zeichenarbeit, die Programme in einer maschinenorientierten Sprache zu schreiben haben (Assemblerprogramme). Auch besteht eher die Möglichkeit, Flussdiagramme durch einen Computer zeichnen zu lassen. Das Flussdiagramm des Euklidischen Algorithmus wurde hier in beiden Darstellungsformen abgebildet, damit der Leser den unterschiedlichen Zeichenaufwand vergleichen kann. Bild 3 zeigt die übliche DIN-Symbolik, Bild 4 den Vorschlag einer vereinfachten Flussdiagrammdarstellung.
Aus diesen Diagrammen ist ersichtlich, dass die Algorithmusstruktur nach Bild 1 mit folgenden Operationsgruppen erfüllt ist:

- Bereitstellen der Eingangsgrössen,
- System der Regeln und Rechenvorschriften,
- Entscheidungsregel,
- Ausgeben der definitiven Ausgangsgrösse.